

CPU12RM/AD  
REV. 1

# 8HC12

## CPU12

REFERENCE  
MANUAL



**MOTOROLA**

# CPU12

## REFERENCE MANUAL

Motorola reserves the right to make changes without further notice to any products herein. Motorola makes no warranty, representation or guarantee regarding the suitability of its products for any particular purpose, nor does Motorola assume any liability arising out of the application or use of any product or circuit, and specifically disclaims any and all liability, including without limitation consequential or incidental damages. "Typical" parameters can and do vary in different applications. All operating parameters, including "Typicals" must be validated for each customer application by customer's technical experts. Motorola does not convey any license under its patent rights nor the rights of others. Motorola products are not designed, intended, or authorized for use as components in systems intended for surgical implant into the body, or other applications intended to support or sustain life, or for any other application in which the failure of the Motorola product could create a situation where personal injury or death may occur. Should Buyer purchase or use Motorola products for any such unintended or unauthorized application, Buyer shall indemnify and hold Motorola and its officers, employees, subsidiaries, affiliates, and distributors harmless against all claims, costs, damages, and expenses, and reasonable attorney fees arising out of, directly or indirectly, any claim of personal injury or death associated with such unintended or unauthorized use, even if such claim alleges that Motorola was negligent regarding the design or manufacture of the part.



MOTOROLA is a registered trademark of Motorola, Inc. Motorola, Inc. is an Equal Opportunity/Affirmative Action Employer.





# TABLE OF CONTENTS

Paragraph		Page
<b>SECTION 1</b>		
<b>INTRODUCTION</b>		
1.1	CPU12 Features .....	1-1
1.2	Readership.....	1-1
1.3	Symbols and Notation .....	1-2
<b>SECTION 2</b>		
<b>OVERVIEW</b>		
2.1	Programming Model.....	2-1
2.2	Data Types.....	2-5
2.3	Memory Organization.....	2-5
2.4	Instruction Queue.....	2-5
<b>SECTION 3</b>		
<b>ADDRESSING MODES</b>		
3.1	Mode Summary.....	3-1
3.2	Effective Address .....	3-2
3.3	Inherent Addressing Mode .....	3-2
3.4	Immediate Addressing Mode .....	3-2
3.5	Direct Addressing Mode.....	3-3
3.6	Extended Addressing Mode .....	3-3
3.7	Relative Addressing Mode .....	3-4
3.8	Indexed Addressing Modes.....	3-5
3.9	Instructions Using Multiple Modes .....	3-10
3.10	Addressing More than 64 Kbytes.....	3-12
<b>SECTION 4</b>		
<b>INSTRUCTION QUEUE</b>		
4.1	Queue Description .....	4-1
4.2	Data Movement in the Queue .....	4-2
4.3	Changes in Execution Flow.....	4-2
<b>SECTION 5</b>		
<b>INSTRUCTION SET OVERVIEW</b>		
5.1	Instruction Set Description .....	5-1
5.2	Load and Store Instructions .....	5-1
5.3	Transfer and Exchange Instructions .....	5-2
5.4	Move Instructions.....	5-3
5.5	Addition and Subtraction Instructions.....	5-3
5.6	Binary Coded Decimal Instructions .....	5-4
5.7	Decrement and Increment Instructions .....	5-4
5.8	Compare and Test Instructions .....	5-5
5.9	Boolean Logic Instructions .....	5-6
5.10	Clear, Complement, and Negate Instructions .....	5-6

## TABLE OF CONTENTS

Paragraph	Page
5.11 Multiplication and Division Instructions .....	5-7
5.12 Bit Test and Manipulation Instructions .....	5-7
5.13 Shift and Rotate Instructions .....	5-8
5.14 Fuzzy Logic Instructions.....	5-9
5.15 Maximum and Minimum Instructions.....	5-11
5.16 Multiply and Accumulate Instruction.....	5-11
5.17 Table Interpolation Instructions .....	5-12
5.18 Branch Instructions .....	5-13
5.19 Loop Primitive Instructions .....	5-16
5.20 Jump and Subroutine Instructions.....	5-17
5.21 Interrupt Instructions .....	5-18
5.22 Index Manipulation Instructions.....	5-19
5.23 Stacking Instructions .....	5-20
5.24 Pointer and Index Calculation Instructions.....	5-20
5.25 Condition Code Instructions .....	5-21
5.26 STOP and WAIT Instructions .....	5-21
5.27 Background Mode and Null Operations .....	5-22

### SECTION 6 INSTRUCTION GLOSSARY

6.1 Glossary Information .....	6-1
6.2 Condition Code Changes .....	6-2
6.3 Object Code Notation.....	6-2
6.4 Source Forms.....	6-3
6.5 Cycle-by-Cycle Execution .....	6-5
6.6 Glossary .....	6-8

### SECTION 7 EXCEPTION PROCESSING

7.1 Types of Exceptions.....	7-1
7.2 Exception Priority .....	7-2
7.3 Resets .....	7-2
7.4 Interrupts .....	7-3
7.5 Unimplemented Opcode Trap .....	7-5
7.6 Software Interrupt Instruction .....	7-6
7.7 Exception Processing Flow .....	7-6

### SECTION 8 DEVELOPMENT AND DEBUG SUPPORT

8.1 External Reconstruction of the Queue .....	8-1
8.2 Instruction Queue Status Signals.....	8-1
8.3 Implementing Queue Reconstruction.....	8-3
8.4 Background Debug Mode .....	8-6
8.5 Instruction Tagging.....	8-13

## TABLE OF CONTENTS

Paragraph	Page
8.6 Breakpoints .....	8-14
<b>SECTION 9</b>	
<b>FUZZY LOGIC SUPPORT</b>	
9.1 Introduction .....	9-1
9.2 Fuzzy Logic Basics .....	9-1
9.3 Example Inference Kernel .....	9-7
9.4 MEM Instruction Details .....	9-9
9.5 REV, REVW Instruction Details .....	9-13
9.6 WAV Instruction Details .....	9-22
9.7 Custom Fuzzy Logic Programming .....	9-26
<b>SECTION 10</b>	
<b>MEMORY EXPANSION</b>	
10.1 Expansion System Description .....	10-1
10.2 CALL and Return from Call Instructions .....	10-3
10.3 Address Lines for Expansion Memory .....	10-4
10.4 Overlay Window Controls .....	10-4
10.5 Using Chip-Select Circuits .....	10-5
10.6 System Notes .....	10-7
<b>APPENDIX A</b>	
<b>INSTRUCTION REFERENCE</b>	
A.1 Instruction Set Summary .....	A-1
A.2 Opcode Map .....	A-1
A.3 Indexed Addressing Postbyte Encoding .....	A-1
A.4 Transfer and Exchange Postbyte Encoding .....	A-1
A.5 Loop Primitive Postbyte Encoding .....	A-1
<b>APPENDIX B</b>	
<b>M68HC11 TO M68HC12 UPGRADE PATH</b>	
B.1 CPU12 Design Goals .....	B-1
B.2 Source Code Compatibility .....	B-1
B.3 Programmer's Model and Stacking .....	B-3
B.4 True 16-Bit Architecture .....	B-3
B.5 Improved Indexing .....	B-6
B.6 Improved Performance .....	B-9
B.7 Additional Functions .....	B-11
<b>APPENDIX C</b>	
<b>HIGH-LEVEL LANGUAGE SUPPORT</b>	
C.1 Data Types .....	C-1
C.2 Parameters and Variables .....	C-1
C.3 Increment and Decrement Operators .....	C-3

## TABLE OF CONTENTS

Paragraph	Page
C.4 Higher Math Functions.....	C-3
C.5 Conditional If Constructs.....	C-4
C.6 Case and Switch Statements.....	C-4
C.7 Pointers.....	C-4
C.8 Function Calls .....	C-4
C.9 Instruction Set Orthogonality.....	C-5

### APPENDIX D ASSEMBLY LISTING

#### INDEX

#### SUMMARY OF CHANGES

### APPENDIX A INSTRUCTION REFERENCE

A.1 Instruction Set Summary.....	A-1
A.2 Opcode Map.....	A-2
A.3 Instruction Addressing Modes.....	A-3
A.4 Transfer and Exchange Instructions.....	A-4
A.5 Loop Instructions.....	A-5

### APPENDIX B MACHINE TO MACHINE UPGRADE PATH

B.1 CPU12 Design Goals.....	B-1
B.2 Source Code Compatibility.....	B-2
B.3 Programmer's Model and Syntax.....	B-3
B.4 The 16-Bit Architecture.....	B-4
B.5 Improved Instruction.....	B-5
B.6 Improved Performance.....	B-6
B.7 Addressed Functions.....	B-7

### APPENDIX C HIGH-LEVEL LANGUAGE SUPPORT

C.1 Data Types.....	C-1
C.2 Parameters and Variables.....	C-2
C.3 Statements and Expression Operators.....	C-3

## LIST OF ILLUSTRATIONS

Figure	Page
2-1 Programming Model.....	2-1
6-1 Example Glossary Page.....	6-1
7-2 Exception Processing Flow Diagram .....	7-7
8-1 Queue Status Signal Timing .....	8-2
8-2 BDM Host to Target Serial Bit Timing .....	8-8
8-3 BDM Target to Host Serial Bit Timing (Logic 1) .....	8-8
8-4 BDM Target to Host Serial Bit Timing (Logic 0) .....	8-9
8-5 Tag Input Timing .....	8-13
9-1 Block Diagram of a Fuzzy Logic System.....	9-3
9-2 Fuzzification Using Membership Functions.....	9-4
9-3 Fuzzy Inference Engine .....	9-8
9-4 Defining a Normal Membership Function.....	9-10
9-5 MEM Instruction Flow Diagram .....	9-11
9-6 Abnormal Membership Function Case 1 .....	9-12
9-7 Abnormal Membership Function Case 2.....	9-13
9-8 Abnormal Membership Function Case 3.....	9-13
9-9 REV Instruction Flow Diagram .....	9-16
9-10 REVW Instruction Flow Diagram.....	9-21
9-11 WAV and wavr Instruction Flow Diagram.....	9-25
9-12 Endpoint Table Handling.....	9-28

# LIST OF ILLUSTRATIONS

Page	Figure
2-1	Programming Model
2-1	Example Glossary Page
2-2	Exception Processing Flow Diagram
2-3	Queue Status Signal Timing
2-4	BDM Host to Target Serial GM Timing
2-5	BDM Target to Host Serial GM Timing (Logic 1)
2-6	BDM Target to Host Serial GM Timing (Logic 0)
2-7	Tag Inlet Timing
2-8	Block Diagram of a Fuzzy Logic System
2-9	Fuzzification Using Membership Functions
2-10	Fuzzy Inference Engine
2-11	Defining a Normal Membership Function
2-12	MBM Inlet and Flow Diagram
2-13	Abnormal Membership Function Case 1
2-14	Abnormal Membership Function Case 2
2-15	Abnormal Membership Function Case 3
2-16	BEV Instruction Flow Diagram
2-17	BEV Instruction Flow Diagram
2-18	WAV and wav Instruction Flow Diagram
2-19	Exception Test Handling



## LIST OF TABLES

Table	Page
3-1 M68HC12 Addressing Mode Summary.....	3-1
3-2 Summary of Indexed Operations .....	3-6
3-3 PC Offsets for Move Instructions .....	3-11
5-1 Load and Store Instructions .....	5-2
5-2 Transfer and Exchange Instructions .....	5-3
5-3 Move Instructions .....	5-3
5-4 Addition and Subtraction Instructions.....	5-4
5-5 BCD Instructions .....	5-4
5-6 Decrement and Increment Instructions .....	5-5
5-7 Compare and Test Instructions .....	5-5
5-8 Boolean Logic Instructions .....	5-6
5-9 Clear, Complement, and Negate Instructions .....	5-6
5-10 Multiplication and Division Instructions .....	5-7
5-11 Bit Test and Manipulation Instructions .....	5-7
5-12 Shift and Rotate Instructions .....	5-8
5-13 Fuzzy Logic Instructions.....	5-10
5-14 Minimum and Maximum Instructions.....	5-11
5-15 Multiply and Accumulate Instructions.....	5-12
5-16 Table Interpolation Instructions.....	5-12
5-17 Short Branch Instructions.....	5-14
5-18 Long Branch Instructions .....	5-15
5-19 Bit Condition Branch Instructions.....	5-16
5-20 Loop Primitive Instructions.....	5-16
5-21 Jump and Subroutine Instructions.....	5-17
5-22 Interrupt Instructions .....	5-18
5-23 Index Manipulation Instructions.....	5-19
5-24 Stacking Instructions.....	5-20
5-25 Pointer and Index Calculation Instructions.....	5-21
5-26 Condition Codes Instructions .....	5-21
5-27 STOP and WAIT Instructions .....	5-22
5-28 Background Mode and Null Operation Instructions.....	5-22
7-1 CPU12 Exception Vector Map .....	7-1
7-2 Stacking Order on Entry to Interrupts.....	7-5
8-1 IPIPE[1:0] Decoding.....	8-2
8-2 BDM Commands Implemented in Hardware.....	8-10
8-3 BDM Firmware Commands.....	8-11
8-4 BDM Register Mapping .....	8-11
8-5 Tag Pin Function .....	8-13
10-1 Mapping Precedence .....	10-2
A-1 Instruction Set Summary.....	A-2
A-2 CPU12 Opcode Map.....	A-20
A-3 Indexed Addressing Mode Summary.....	A-22
A-4 Indexed Addressing Mode Postbyte Encoding (xb) .....	A-23

## LIST OF TABLES

A-5	Transfer and Exchange Postbyte Encoding.....	A-24
A-6	Loop Primitive Postbyte Encoding (lb) .....	A-25
B-1	Translated M68HC11 Mnemonics.....	B-2
B-2	Instructions with Smaller Object Code .....	B-3
B-3	Comparison of Math Instruction Speeds.....	B-10
B-4	New M68HC12 Instructions .....	B-11

## SECTION 1 INTRODUCTION

This manual describes the features and operation of the CPU12 processing unit used in all M68HC12 microcontrollers.

### 1.1 CPU12 Features

The CPU12 is a high-speed, 16-bit processing unit that has a programming model identical to that of the industry standard M68HC11 CPU. The CPU12 instruction set is a proper superset of the M68HC11 instruction set, and M68HC11 source code is accepted by CPU12 assemblers with no changes.

The CPU12 has full 16-bit data paths and can perform arithmetic operations up to 20 bits wide for high-speed math execution.

Unlike many other 16-bit CPUs, the CPU12 allows instructions with odd byte counts, including many single-byte instructions. This allows much more efficient use of ROM space.

An instruction queue buffers program information so the CPU has immediate access to at least three bytes of machine code at the start of every instruction.

In addition to the addressing modes found in other Motorola MCUs, the CPU12 offers an extensive set of indexed addressing capabilities including:

- Using the stack pointer as an index register in all indexed operations
- Using the program counter as an index register in all but auto inc/dec mode
- Accumulator offsets allowed using A, B, or D accumulators
- Automatic pre- or post-increment or pre- or post-decrement (by -8 to +8)
- 5-bit, 9-bit, or 16-bit signed constant offsets
- 16-bit offset indexed-indirect and accumulator D offset indexed-indirect addressing

### 1.2 Readership

This manual is written for professionals and students in electronic design and software development. The primary goal is to provide information necessary to implement control systems using M68HC12 devices. Basic knowledge of electronics, microprocessors, and assembly language programming is required to use the manual effectively. Because the CPU12 has a great deal of commonality with the M68HC11 CPU, prior knowledge of M68HC11 devices is helpful, but is not essential. The CPU12 also includes features that are new and unique. In these cases, there is supplementary material in the text to explain the new technology.

### 1.3 Symbols and Notation

The following symbols and notation are used throughout the manual. More specialized usages that apply only to the instruction glossary are described at the beginning of that section.

#### 1.3.1 Abbreviations for System Resources

- A — Accumulator A
- B — Accumulator B
- D — Double accumulator D (A : B)
- X — Index register X
- Y — Index register Y
- SP — Stack pointer
- PC — Program counter
- CCR — Condition code register
  - S — STOP instruction control bit
  - X — Non-maskable interrupt control bit
  - H — Half-carry status bit
  - I — Maskable interrupt control bit
  - N — Negative status bit
  - Z — Zero status bit
  - V — Two's complement overflow status bit
  - C — Carry/Borrow status bit

#### 1.3.2 Memory and Addressing

- M — 8-bit memory location pointed to by the effective address of the instruction
- M : M+1 — 16-bit memory location. Consists of the location pointed to by the effective address concatenated with the next higher memory location. The most significant byte is at location M.
- M~M+3 — 32-bit memory location. Consists of the effective address of the instruction concatenated with the next three higher memory locations. The most significant byte is at location M or M(Y).
- M(Y)~M(Y+3) — Memory locations pointed to by index register Y
- M(X) — Memory locations pointed to by index register X
- M(SP) — Memory locations pointed to by the stack pointer
- M(Y+3) — Memory locations pointed to by index register Y plus 3, respectively.
- PPAGE — Program overlay page (bank) number for extended memory (>64K).
- Page — Program overlay page
  - X<sub>H</sub> — High-order byte
  - X<sub>L</sub> — Low-order byte
  - ( ) — Content of register or memory location
  - \$ — Hexadecimal value
  - % — Binary value

### 1.3.3 Operators

- + — Addition
- — Subtraction
- — Logical AND
- + — Logical OR (inclusive)
- ⊕ — Logical exclusive OR
- × — Multiplication
- ÷ — Division
- $\overline{M}$  — Negation. One's complement (invert each bit of M)
- : — Concatenate  
Example: A : B means: "The 16-bit value formed by concatenating 8-bit accumulator A with 8-bit accumulator B."  
A is in the high order position.
- ⇒ — Transfer  
Example: (A) ⇒ M means: "The content of accumulator A is transferred to memory location M."
- ↔ — Exchange  
Example: D ↔ X means: "Exchange the contents of D with those of X."

### 1.3.4 Conventions

**Logic level one** is the voltage that corresponds to the True (1) state.

**Logic level zero** is the voltage that corresponds to the False (0) state.

**Set** refers specifically to establishing logic level one on a bit or bits.

**Cleared** refers specifically to establishing logic level zero on a bit or bits.

**Asserted** means that a signal is in active logic state. An active low signal changes from logic level one to logic level zero when asserted, and an active high signal changes from logic level zero to logic level one.

**Negated** means that an asserted signal changes logic state. An active low signal changes from logic level zero to logic level one when negated, and an active high signal changes from logic level one to logic level zero.

**ADDR** is the mnemonic for address bus.

**DATA** is the mnemonic for data bus.

**LSB** means least significant bit or bits; **MSB**, most significant bit or bits.

**LSW** means least significant word or words; **MSW**, most significant word or words.

**A specific mnemonic** within a range is referred to by mnemonic and number. A7 is bit 7 of accumulator A. **A range of mnemonics** is referred to by mnemonic and the numbers that define the range. DATA[15:8] form the high byte of the data bus.

### 1.3.3 Operators

- + -- Addition
- -- Subtraction
- \* -- Logical AND
- / -- Logical OR (inclusive)
- % -- Logical exclusive OR
- \* -- Multiplication
- / -- Division
- ! -- Negation. One's complement (inverted each bit of M)
- ~ -- Complement
- Example: A : B means: The 16-bit value formed by concatenating 8-bit accumulator A with 8-bit accumulator B
- A is in the high order position
- ~ -- Transfer
- Example: (A) <- M means: "The content of accumulator A is transferred to memory location M."
- <- -- Exchange
- Example: D <- X means: "Exchange the contents of D with the contents of X."

### 1.3.4 Conventions

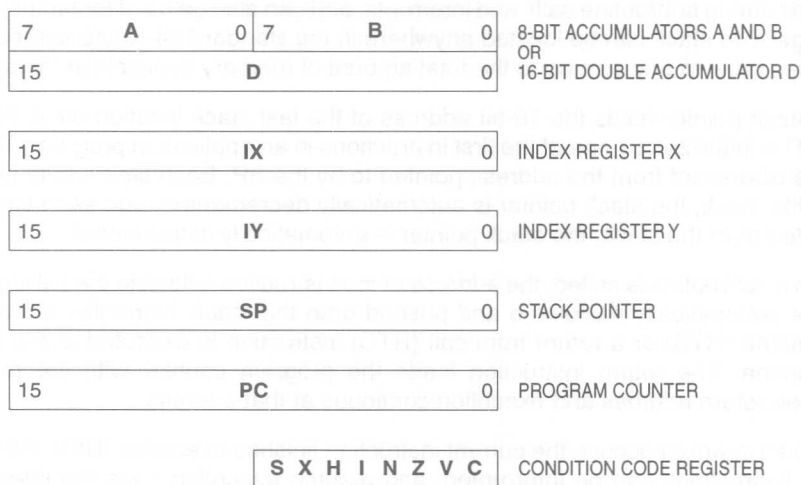
- Logic level one is the voltage that corresponds to the True (1) state.
- Logic level zero is the voltage that corresponds to the False (0) state.
- Set refers specifically to establishing logic level one on a bit or bits.
- Clear refers specifically to establishing logic level zero on a bit or bits.
- Asserted means that a signal is in active logic state. An active low signal being driven from logic level one to logic level zero when asserted and otherwise being held at logic level one.
- Negated means that an asserted signal changes logic state. An active low signal changes from logic level zero to logic level one when negated, and an active high signal changes from logic level one to logic level zero.
- ADDR is the mnemonic for address bus.
- DATA is the mnemonic for data bus.
- LSB means least significant bit or bits; MSB, most significant bit or bits.
- LSW means least significant word or words; MSW, most significant word or words.
- A specific mnemonic within a range is referred to by mnemonic and number. An 8-bit accumulator A range of mnemonics is referred to by mnemonic and bit number that define the range. DATA[7:5] from the high byte of the data bus.

## SECTION 2 OVERVIEW

This section describes the CPU12 programming model, register set, the data types used, and basic memory organization.

### 2.1 Programming Model

The CPU12 programming model, shown in **Figure 2-1**, is the same as that of the M68HC11 CPU. The CPU has two 8-bit general-purpose accumulators (A and B) that can be concatenated into a single 16-bit accumulator (D) for certain instructions. It also has two index registers (X and Y), a 16-bit stack pointer (SP), a 16-bit program counter (PC), and an 8-bit condition code register (CCR).



HC12 PROG MODEL

**Figure 2-1 Programming Model**

#### 2.1.1 Accumulators

General-purpose 8-bit accumulators A and B are used to hold operands and results of operations. Some instructions treat the combination of these two 8-bit accumulators (A : B) as a 16-bit double accumulator (D).



Most operations can use accumulator A or B interchangeably. However, there are a few exceptions. Add, subtract, and compare instructions involving both A and B (ABA, SBA, and CBA) only operate in one direction, so it is important to make certain the correct operand is in the correct accumulator. The decimal adjust accumulator A (DAA) instruction is used after binary-coded decimal (BCD) arithmetic operations. There is no equivalent instruction to adjust accumulator B.

### 2.1.2 Index Registers

16-bit index registers X and Y are used for indexed addressing. In the indexed addressing modes, the contents of an index register are added to 5-bit, 9-bit, or 16-bit constants or to the content of an accumulator to form the effective address of the instruction operand. The second index register is especially useful for moves and in cases where operands from two separate tables are used in a calculation.

### 2.1.3 Stack Pointer

The CPU12 supports an automatic program stack. The stack is used to save system context during subroutine calls and interrupts, and can also be used for temporary data storage. The stack can be located anywhere in the standard 64-Kbyte address space and can grow to any size up to the total amount of memory available in the system.

The stack pointer holds the 16-bit address of the last stack location used. Normally, the SP is initialized by one of the first instructions in an application program. The stack grows downward from the address pointed to by the SP. Each time a byte is pushed onto the stack, the stack pointer is automatically decremented, and each time a byte is pulled from the stack, the stack pointer is automatically incremented.

When a subroutine is called, the address of the instruction following the calling instruction is automatically calculated and pushed onto the stack. Normally, a return from subroutine (RTS) or a return from call (RTC) instruction is executed at the end of a subroutine. The return instruction loads the program counter with the previously stacked return address and execution continues at that address.

When an interrupt occurs, the current instruction finishes execution (REV, REVW, and WAV instructions can be interrupted, and resume execution once the interrupt has been serviced), the address of the next instruction is calculated and pushed onto the stack, all the CPU registers are pushed onto the stack, the program counter is loaded with the address pointed to by the interrupt vector, and execution continues at that address. The stacked registers are referred to as an interrupt stack frame. The CPU12 stack frame is the same as that of the M68HC11.

### 2.1.4 Program Counter

The program counter (PC) is a 16-bit register that holds the address of the next instruction to be executed. It is automatically incremented each time an instruction is fetched.

### 2.1.5 Condition Code Register

This register contains five status indicators, two interrupt masking bits, and a STOP instruction control bit. It is named for the five status indicators.

The status bits reflect the results of CPU operation as it executes instructions. The five flags are half carry (H), negative (N), zero (Z), overflow (V), and carry/borrow (C). The half-carry flag is used only for BCD arithmetic operations. The N, Z, V, and C status bits allow for branching based on the results of a previous operation.

In some architectures, only a few instructions affect condition codes, so that multiple instructions must be executed in order to load and test a variable. Since most CPU12 instructions automatically update condition codes, it is rarely necessary to execute an extra instruction for this purpose. The challenge in using the CPU12 lies in finding instructions that do not alter the condition codes. The most important of these instructions are pushes, pulls, transfers, and exchanges.

It is always a good idea to refer to an instruction set summary (see **APPENDIX A INSTRUCTION REFERENCE**) to check which condition codes are affected by a particular instruction.

The following paragraphs describe normal uses of the condition codes. There are other, more specialized uses. For instance, the C status bit is used to enable weighted fuzzy logic rule evaluation. Specialized usages are described in the relevant portions of this manual and in **SECTION 6 INSTRUCTION GLOSSARY**.

#### 2.1.5.1 S Control Bit

Setting the S bit disables the STOP instruction. Execution of a STOP instruction causes the on-chip oscillator to stop. This may be undesirable in some applications. If the CPU encounters a STOP instruction while the S bit is set, it is treated like a no-operation (NOP) instruction, and continues to the next instruction.

#### 2.1.5.2 X Mask Bit

The  $\overline{\text{XIRQ}}$  input is an updated version of the  $\overline{\text{NMI}}$  input found on earlier generations of MCUs. Non-maskable interrupts are typically used to deal with major system failures, such as loss of power. However, enabling non-maskable interrupts before a system is fully powered and initialized can lead to spurious interrupts. The X bit provides a mechanism for enabling non-maskable interrupts after a system is stable.

By default, the X bit is set to one during reset. As long as the X bit remains set, interrupt service requests made via the  $\overline{\text{XIRQ}}$  pin are not recognized. An instruction must clear the X bit to enable non-maskable interrupt service requests made via the  $\overline{\text{XIRQ}}$  pin. Once the X bit has been cleared to zero, software cannot reset it to one by writing to the CCR. The X bit is not affected by maskable interrupts.

When an  $\overline{\text{XIRQ}}$  interrupt occurs after non-maskable interrupts are enabled, both the X bit and the I bit are automatically set to prevent other interrupts from being recognized during the interrupt service routine. The mask bits are set after the registers are stacked, but before the interrupt vector is fetched.

Normally, an RTI instruction at the end of the interrupt service routine restores register values that were present before the interrupt occurred. Since the CCR is stacked before the X bit is set, the RTI normally clears the X bit, and thus re-enables non-maskable interrupts. While it is possible to manipulate the stacked value of X so that X is set after an RTI, there is no software method to re-set X (and disable  $\overline{\text{NMI}}$ ) once X has been cleared.

#### 2.1.5.3 H Status Bit

The H bit indicates a carry from accumulator A bit 3 during an addition operation. The DAA instruction uses the value of the H bit to adjust a result in accumulator A to correct BCD format. H is updated only by the ABA, ADD, and ADC instructions.

#### 2.1.5.4 I Mask Bit

The I bit enables and disables maskable interrupt sources. By default, the I bit is set to one during reset. An instruction must clear the I bit to enable maskable interrupts. While the I bit is set, maskable interrupts can become pending and are remembered, but operation continues uninterrupted until the I bit is cleared.

When an interrupt occurs after interrupts are enabled, the I bit is automatically set to prevent other maskable interrupts during the interrupt service routine. The I bit is set after the registers are stacked, but before the interrupt vector is fetched.

Normally, an RTI instruction at the end of the interrupt service routine restores register values that were present before the interrupt occurred. Since the CCR is stacked before the I bit is set, the RTI normally clears the I bit, and thus re-enables interrupts. Interrupts can be re-enabled by clearing the I bit within the service routine, but implementing a nested interrupt management scheme requires great care, and seldom improves system performance.

#### 2.1.5.5 N Status Bit

The N bit shows the state of the MSB of the result. N is most commonly used in two's complement arithmetic, where the MSB of a negative number is one and the MSB of a positive number is zero, but it has other uses. For instance, if the MSB of a register or memory location is used as a status flag, the user can test status by loading an accumulator.

#### 2.1.5.6 Z Status Bit

The Z bit is set when all the bits of the result are zeros. Compare instructions perform an internal implied subtraction, and the condition codes, including Z, reflect the results of that subtraction. The INX, DEX, INY, and DEY instructions affect the Z bit and no other condition flags. These operations can only determine = and  $\neq$ .

#### 2.1.5.7 V Status Bit

The V bit is set when two's complement overflow occurs as a result of an operation.

### 2.1.5.8 C Status Bit

The C bit is set when a carry occurs during addition or a borrow occurs during subtraction. The C bit also acts as an error flag for multiply and divide operations. Shift and rotate instructions operate through the C bit to facilitate multiple-word shifts.

## 2.2 Data Types

The CPU12 uses the following types of data:

- Bits
- 5-bit signed integers
- 8-bit signed and unsigned integers
- 8-bit, 2-digit binary coded decimal numbers
- 9-bit signed integers
- 16-bit signed and unsigned integers
- 16-bit effective addresses
- 32-bit signed and unsigned integers

Negative integers are represented in two's complement form.

Five-bit and 9-bit signed integers are used only as offsets for indexed addressing modes.

Sixteen-bit effective addresses are formed during addressing mode computations.

Thirty-two-bit integer dividends are used by extended division instructions. Extended multiply and extended multiply-and-accumulate instructions produce 32-bit products.

## 2.3 Memory Organization

The standard CPU12 address space is 64 Kbytes. Some M68HC12 devices support a paged memory expansion scheme that increases the standard space by means of predefined windows in address space. The CPU12 has special instructions that support use of expanded memory. See **SECTION 10 MEMORY EXPANSION** for more information.

Eight-bit values can be stored at any odd or even byte address in available memory. Sixteen-bit values are stored in memory as two consecutive bytes; the high byte occupies the lowest address, but need not be aligned to an even boundary. Thirty-two-bit values are stored in memory as four consecutive bytes; the high byte occupies the lowest address, but need not be aligned to an even boundary.

All I/O and all on-chip peripherals are memory-mapped. No special instruction syntax is required to access these addresses. On-chip registers and memory are typically grouped in blocks which can be relocated within the standard 64-Kbyte address space. Refer to device documentation for specific information.

## 2.4 Instruction Queue

The CPU12 uses an instruction queue to buffer program information. The mechanism is called a queue rather than a pipeline because a typical pipelined CPU executes more than one instruction at the same time, while the CPU12 always finishes executing an instruction before beginning to execute another. Refer to **SECTION 4 INSTRUCTION QUEUE** for more information.

### 3.1.5.8 C Status Bit

The C bit is set when a carry occurs during addition or a borrow occurs during subtraction. The C bit also acts as an error flag for multiply and divide operations. 5-bit and 16-bit instructions operate through the C bit to facilitate multiply and divide.

### 3.2 Data Types

The CPU12 uses the following types of data:

- 8-bit
- 7-bit signed integers
- 5-bit signed and unsigned integers
- 4-bit, 5-digit binary coded decimal numbers
- 2-bit signed integers
- 16-bit signed and unsigned integers
- 16-bit effective addresses
- 32-bit signed and unsigned integers

Negative integers are represented in two's complement form.

Five-bit and 9-bit signed integers are used only as offsets in branch and testing modes.

Sixteen-bit effective addresses are formed during addressing mode computations.

Thirty-two-bit integer dividends are used by extended division instructions. 5-decimal multiply and extended multiply-and-accumulate instructions produce 32-bit products.

### 3.3 Memory Organization

The standard CPU12 address space is 64 Kbytes. Some 16-bit HLE level as against a page memory expansion scheme that increases the standard space by a factor of 16. The CPU12 has several internal and external predefined windows in address space. See SECTION 10 MEMORY EXPANSION for more details on expanded memory.

Eight-bit values can be stored at any odd or even byte address if available. Eight-bit values are stored in memory as two consecutive bytes, the 4th byte (hex) 1, plus the lowest address, but need not be aligned to an even boundary. The 4th values are stored in memory as four consecutive bytes, the 4th byte (hex) 1, plus the lowest address, but need not be aligned to an even boundary.

All 80 and 16-bit on-chip peripherals are memory-mapped. 16-bit peripheral registers are addressed. On-chip registers are memory-mapped and are 16-bit. A register is required to access these addresses. On-chip registers are memory-mapped and are 16-bit. A register is required to access these addresses. On-chip registers are memory-mapped and are 16-bit. A register is required to access these addresses.

### 3.4 Instruction Queue

The CPU12 uses an instruction queue to buffer program instructions. The instruction queue is a queue rather than a pipeline because a typical program would not execute more than one instruction at the same time, while the CPU12 always has a pipeline. The instruction queue is 16 words long. The instruction queue is 16 words long. The instruction queue is 16 words long.

## SECTION 3 ADDRESSING MODES

Addressing modes determine how the CPU accesses memory locations to be operated upon. This section discusses the various modes and how they are used.

### 3.1 Mode Summary

Addressing modes are an implicit part of CPU12 instructions. **APPENDIX A INSTRUCTION REFERENCE** shows the modes used by each instruction. All CPU12 addressing modes are shown in **Table 3-1**.

**Table 3-1 M68HC12 Addressing Mode Summary**

Addressing Mode	Source Format	Abbreviation	Description
Inherent	<b>INST</b> (no externally supplied operands)	INH	Operands (if any) are in CPU registers
Immediate	<b>INST #opr8i</b> or <b>INST #opr16i</b>	IMM	Operand is included in instruction stream 8- or 16-bit size implied by context
Direct	<b>INST opr8a</b>	DIR	Operand is the lower 8-bits of an address in the range \$0000 – \$00FF
Extended	<b>INST opr16a</b>	EXT	Operand is a 16-bit address
Relative	<b>INST rel8</b> or <b>INST rel16</b>	REL	An 8-bit or 16-bit relative offset from the current pc is supplied in the instruction
Indexed (5-bit offset)	<b>INST oprx5,xysp</b>	IDX	5-bit signed constant offset from x, y, sp, or pc
Indexed (pre-decrement)	<b>INST oprx3,-xys</b>	IDX	Auto pre-decrement x, y, or sp by 1 ~ 8
Indexed (pre-increment)	<b>INST oprx3,+xys</b>	IDX	Auto pre-increment x, y, or sp by 1 ~ 8
Indexed (post-decrement)	<b>INST oprx3,xys-</b>	IDX	Auto post-decrement x, y, or sp by 1 ~ 8
Indexed (post-increment)	<b>INST oprx3,xys+</b>	IDX	Auto post-increment x, y, or sp by 1 ~ 8
Indexed (accumulator offset)	<b>INST abd,xysp</b>	IDX	Indexed with 8-bit (A or B) or 16-bit (D) accumulator offset from x, y, sp, or pc
Indexed (9-bit offset)	<b>INST oprx9,xysp</b>	IDX1	9-bit signed constant offset from x, y, sp, or pc (lower 8-bits of offset in one extension byte)
Indexed (16-bit offset)	<b>INST oprx16,xysp</b>	IDX2	16-bit constant offset from x, y, sp, or pc (16-bit offset in two extension bytes)
Indexed-Indirect (16-bit offset)	<b>INST [oprx16,xysp]</b>	[IDX2]	Pointer to operand is found at... 16-bit constant offset from x, y, sp, or pc (16-bit offset in two extension bytes)
Indexed-Indirect (D accumulator offset)	<b>INST [D,xysp]</b>	[D,IDX]	Pointer to operand is found at... x, y, sp, or pc plus the value in D

The CPU12 uses all M68HC11 modes as well as new forms of indexed addressing. Differences between M68HC11 and M68HC12 indexed modes are described in **3.8 Indexed Addressing Modes**. Instructions that use more than one mode are discussed in **3.9 Instructions Using Multiple Modes**.

### 3.2 Effective Address

Each addressing mode except inherent mode generates a 16-bit effective address which is used during the memory reference portion of the instruction. Effective address computations do not require extra execution cycles.

### 3.3 Inherent Addressing Mode

Instructions that use this addressing mode either have no operands or all operands are in internal CPU registers. In either case, the CPU does not need to access any memory locations to complete the instruction.

Examples:

NOP	;this instruction has no operands
INX	;operand is a CPU register

### 3.4 Immediate Addressing Mode

Operands for immediate mode instructions are included in the instruction stream, and are fetched into the instruction queue one 16-bit word at a time during normal program fetch cycles. Since program data is read into the instruction queue several cycles before it is needed, when an immediate addressing mode operand is called for by an instruction, it is already present in the instruction queue.

The pound symbol (#) is used to indicate an immediate addressing mode operand. One very common programming error is to accidentally omit the # symbol. This causes the assembler to misinterpret the following expression as an address rather than explicitly provided data. For example LDAA #\$55 means to load the immediate value \$55 into the A accumulator, while LDAA \$55 means to load the value from address \$0055 into the A accumulator. Without the # symbol the instruction is erroneously interpreted as a direct addressing mode instruction.

Examples:

LDAA	#\$55
LDX	#\$1234
LDY	#\$67

These are common examples of 8-bit and 16-bit immediate addressing mode. The size of the immediate operand is implied by the instruction context. In the third example, the instruction implies a 16-bit immediate value but only an 8-bit value is supplied. In this case the assembler will generate the 16-bit value \$0067 because the CPU expects a 16-bit value in the instruction stream.

BRSET	FOO, #03, THERE
-------	-----------------



In this example, extended addressing mode is used to access the operand FOO, immediate addressing mode is used to access the mask value \$03, and relative addressing mode is used to identify the destination address of a branch in case the branch-taken conditions are met. BRSET is listed as an extended mode instruction even though immediate and relative modes are also used.

### 3.5 Direct Addressing Mode

This addressing mode is sometimes called zero-page addressing because it is used to access operands in the address range \$0000 through \$00FF. Since these addresses always begin with \$00, only the eight low-order bits of the address need to be included in the instruction, which saves program space and execution time. A system can be optimized by placing the most commonly accessed data in this area of memory. The eight low-order bits of the operand address are supplied with the instruction and the eight high-order bits of the address are assumed to be zero.

Examples:

LDAA          \$55

This is a very basic example of direct addressing. The value \$55 is taken to be the low-order half of an address in the range \$0000 through \$00FF. The high order half of the address is assumed to be zero. During execution of this instruction, the CPU combines the value \$55 from the instruction with the assumed value of \$00 to form the address \$0055, which is then used to access the data to be loaded into accumulator A.

LDX          \$20

In this example, the value \$20 is combined with the assumed value of \$00 to form the address \$0020. Since the LDX instruction requires a 16-bit value, a 16-bit word of data is read from addresses \$0020 and \$0021. After execution of this instruction, the X index register will have the value from address \$0020 in its high-order half and the value from address \$0021 in its low-order half.

### 3.6 Extended Addressing Mode

In this addressing mode, the full 16-bit address of the memory location to be operated on is provided in the instruction. This addressing mode can be used to access any location in the 64-Kbyte memory map.

Example:

LDAA          \$F03B

This is a very basic example of extended addressing. The value from address \$F03B is loaded into the A accumulator.

### 3.7 Relative Addressing Mode

The relative addressing mode is used only by branch instructions. Short and long conditional branch instructions use relative addressing mode exclusively, but branching versions of bit manipulation instructions (BRSET and BRCLR) use multiple addressing modes, including relative mode. Refer to **3.9 Instructions Using Multiple Modes** for more information.

Short branch instructions consist of an 8-bit opcode and a signed 8-bit offset contained in the byte that follows the opcode. Long branch instructions consist of an 8-bit pre-byte, an 8-bit opcode and a signed 16-bit offset contained in the two bytes that follow the opcode.

Each conditional branch instruction tests certain status bits in the condition code register. If the bits are in a specified state, the offset is added to the address of the next memory location after the offset to form an effective address, and execution continues at that address; if the bits are not in the specified state, execution continues with the instruction immediately following the branch instruction.

Bit-condition branches test whether bits in a memory byte are in a specific state. Various addressing modes can be used to access the memory location. An 8-bit mask operand is used to test the bits. If each bit in memory that corresponds to a one in the mask is either set (BRSET) or clear (BRCLR), an 8-bit offset is added to the address of the next memory location after the offset to form an effective address, and execution continues at that address; if all the bits in memory that correspond to a one in the mask are not in the specified state, execution continues with the instruction immediately following the branch instruction.

Both 8-bit and 16-bit offsets are signed two's complement numbers to support branching upward and downward in memory. The numeric range of short branch offset values is \$80 (-128) to \$7F (127). The numeric range of long branch offset values is \$8000 (-32768) to \$7FFF (32767). If the offset is zero, the CPU executes the instruction immediately following the branch instruction, regardless of the test involved.

Since the offset is at the end of a branch instruction, using a negative offset value can cause the PC to point to the opcode and initiate a loop. For instance, a branch always (BRA) instruction consists of two bytes, so using an offset of \$FE sets up an infinite loop; the same is true of a long branch always (LBRA) instruction with an offset of \$FFFC.

An offset that points to the opcode can cause a bit-condition branch to repeat execution until the specified bit condition is satisfied. Since bit condition branches can consist of four, five, or six bytes depending on the addressing mode used to access the byte in memory, the offset value that sets up a loop can vary. For instance, using an offset of \$FC with a BRCLR that accesses memory using an 8-bit indexed postbyte sets up a loop that executes until all the bits in the specified memory byte that correspond to ones in the mask byte are cleared.

### 3.8 Indexed Addressing Modes

The CPU12 uses redefined versions of M68HC11 indexed modes that reduce execution time and eliminate code size penalties for using the Y index register. In most cases, CPU12 code size for indexed operations is the same or is smaller than that for the M68HC11. Execution time is shorter in all cases. Execution time improvements are due to both a reduced number of cycles for all indexed instructions and to faster system clock speed.

The indexed addressing scheme uses a postbyte plus 0, 1, or 2 extension bytes after the instruction opcode. The postbyte and extensions do the following tasks:

1. Specify which index register is used.
2. Determine whether a value in an accumulator is used as an offset.
3. Enable automatic pre or post increment or decrement.
4. Specify size of increment or decrement.
5. Specify use of 5-, 9-, or 16-bit signed offsets.

This approach eliminates the differences between X and Y register use while dramatically enhancing the indexed addressing capabilities.

Major advantages of the CPU12 indexed addressing scheme are:

- The stack pointer can be used as an index register in all indexed operations.
- The program counter can be used as an index register in all but autoincrement and autodecrement modes.
- A, B, or D accumulators can be used for accumulator offsets.
- Automatic pre- or post-increment or pre- or post-decrement by -8 to +8
- A choice of 5-, 9-, or 16-bit signed constant offsets.
- Use of two new indexed-indirect modes.
  - Indexed-indirect mode with 16-bit offset
  - Indexed-indirect mode with accumulator D offset

**Table 3-2** is a summary of indexed addressing mode capabilities and a description of postbyte encoding. The postbyte is noted as xb in instruction descriptions. Detailed descriptions of the indexed addressing mode variations follow the table.

All indexed addressing modes use a 16-bit CPU register and additional information to create an effective address. In most cases the effective address specifies the memory location affected by the operation. In some variations of indexed addressing, the effective address specifies the location of a value that points to the memory location affected by the operation.

Indexed addressing mode instructions use a postbyte to specify X, Y, SP, or PC as the base index register and to further classify the way the effective address is formed. A special group of instructions (LEAS, LEAX, and LEAY) cause this calculated effective address to be loaded into an index register for further calculations.

Table 3-2 Summary of Indexed Operations

Postbyte Code (xb)	Source Code Syntax	Comments
		rr; 00 = X, 01 = Y, 10 = SP, 11 = PC
rr0nnnnn	,r n,r -n,r	<b>5-bit constant offset</b> n = -16 to +15 r can specify X, Y, SP, or PC
111rr0zs	n,r -n,r	<b>Constant offset</b> (9- or 16-bit signed) z- 0 = 9-bit with sign in LSB of postbyte(s) -256 < n < 255 1 = 16-bit 0 < n < 65,535 if z = s = 1, 16-bit offset indexed-indirect (see below) r can specify X, Y, SP, or PC
111rr011	[n,r]	<b>16-bit offset indexed-indirect</b> rr can specify X, Y, SP, or PC 0 < n < 65,535
rrlpnnnn	n,-r n,+r n,r- n,r+	<b>Auto pre-decrement/increment or Auto post-decrement/increment;</b> p = pre-(0) or post-(1), n = -8 to -1, +1 to +8 r can specify X, Y, or SP (PC not a valid choice) +8 = 0111 ... +1 = 0000 -1 = 1111 ... -8 = 1000
111rr1aa	A,r B,r D,r	<b>Accumulator offset</b> (unsigned 8-bit or 16-bit) aa- 00 = A 01 = B 10 = D (16-bit) 11 = see accumulator D offset indexed-indirect r can specify X, Y, SP, or PC
111rr111	[D,r]	<b>Accumulator D offset indexed-indirect</b> r can specify X, Y, SP, or PC

### 3.8.1 5-Bit Constant Offset Indexed Addressing

This indexed addressing mode uses a 5-bit signed offset which is included in the instruction postbyte. This short offset is added to the base index register (X, Y, SP, or PC) to form the effective address of the memory location that will be affected by the instruction. This gives a range of -16 through +15 from the value in the base index register. Although other indexed addressing modes allow 9- or 16-bit offsets, those modes also require additional extension bytes in the instruction for this extra information. The majority of indexed instructions in real programs use offsets that fit in the shortest 5-bit form of indexed addressing.

Examples:

```
LDAA      0,X
STAB      -8,Y
```

For these examples, assume X has a value of \$1000 and Y has a value of \$2000 before execution. The 5-bit constant offset mode does not change the value in the index register, so X will still be \$1000 and Y will still be \$2000 after execution of these instructions. In the first example, A will be loaded with the value from address \$1000. In the second example, the value from the B accumulator will be stored at address \$1FF8 (\$2000 - \$8).

### 3.8.2 9-Bit Constant Offset Indexed Addressing

This indexed addressing mode uses a 9-bit signed offset which is added to the base index register (X, Y, SP, or PC) to form the effective address of the memory location affected by the instruction. This gives a range of -256 through +255 from the value in the base index register. The most significant bit (sign bit) of the offset is included in the instruction postbyte and the remaining eight bits are provided as an extension byte after the instruction postbyte in the instruction flow.

Examples:

```
LDAA      $FF,X
```

```
LDAB      -20,Y
```

For these examples assume X is \$1000 and Y is \$2000 before execution of these instructions. (These instructions do not alter the index registers so they will still be \$1000 and \$2000 respectively after the instructions.) The first instruction will load A with the value from address \$10FF and the second instruction will load B with the value from address \$1FEC.

This variation of the indexed addressing mode in the CPU12 is similar to the M68HC11 indexed addressing mode, but is functionally enhanced. The M68HC11 CPU provides for unsigned 8-bit constant offset indexing from X or Y, and use of Y requires an extra instruction byte and thus, an extra execution cycle. The 9-bit signed offset used in the CPU12 covers the same range of positive offsets as the M68HC11, and adds negative offset capability. The CPU12 can use X, Y, SP or PC as the base index register.

### 3.8.3 16-Bit Constant Offset Indexed Addressing

This indexed addressing mode uses a 16-bit offset which is added to the base index register (X, Y, SP, or PC) to form the effective address of the memory location affected by the instruction. This allows access to any address in the 64-Kbyte address space. Since the address bus and the offset are both 16 bits, it does not matter whether the offset value is considered to be a signed or an unsigned value (\$FFFF may be thought of as +65,535 or as -1). The 16-bit offset is provided as two extension bytes after the instruction postbyte in the instruction flow.

### 3.8.4 16-Bit Constant Indirect Indexed Addressing

This indexed addressing mode adds a 16-bit instruction-supplied offset to the base index register to form the address of a memory location that contains a pointer to the memory location affected by the instruction. The instruction itself does not point to the address of the memory location to be acted upon, but rather to the location of a pointer to the address to be acted on. The square brackets distinguish this addressing mode from 16-bit constant offset indexing.

Example:

```
LDAA      [10,X]
```

In this example, X holds the base address of a table of pointers. Assume that X has an initial value of \$1000, and that the value \$2000 is stored at addresses \$100A and \$100B. The instruction first adds the value 10 to the value in X to form the address \$100A. Next, an address pointer (\$2000) is fetched from memory at \$100A. Then, the value stored in location \$2000 is read and loaded into the A accumulator.

### 3.8.5 Auto Pre/Post Decrement/Increment Indexed Addressing

This indexed addressing mode provides four ways to automatically change the value in a base index register as a part of instruction execution. The index register can be incremented or decremented by an integer value either before or after indexing takes place. The base index register may be X, Y, or SP (auto-modify modes would not make sense on PC).

Pre decrement and pre increment versions of the addressing mode adjust the value of the index register before accessing the memory location affected by the instruction — the index register retains the changed value after the instruction executes. Post-decrement and post-increment versions of the addressing mode use the initial value in the index register to access the memory location affected by the instruction, then change the value of the index register.

The CPU12 allows the index register to be incremented or decremented by any integer value in the ranges  $-8$  through  $-1$ , or  $1$  through  $8$ . The value need not be related to the size of the operand for the current instruction. These instructions can be used to incorporate an index adjustment into an existing instruction rather than using an additional instruction and increasing execution time. This addressing mode is also used to perform operations on a series of data structures in memory.

When an LEAS, LEAX, or LEAY instruction is executed using this addressing mode, and the operation modifies the index register that is being loaded, the final value in the register is the value that would have been used to access a memory operand (premodification is seen in the result but postmodification is not).

Examples:

STAA	1,-SP	;equivalent to PSHA
STX	2,-SP	;equivalent to PSHX
LDX	2,SP+	;equivalent to PULX
LDAA	1,SP+	;equivalent to PULA

For a “last-used” type of stack like the CPU12 stack, these four examples are equivalent to common push and pull instructions. For a “next-available” stack like the M68HC11 stack, PSHA is equivalent to STAA 1,SP- and PULA is equivalent to LDAA 1,SP+. However, in the M68HC11, 16-bit operations like PSHX and PULX require multiple instructions to decrement the SP by one, then store X, then decrement SP by one again.

In the STAA 1,-SP example, the stack pointer is pre-decremented by one and then A is stored to the address contained in the stack pointer. Similarly the LDX 2,SP+ first loads X from the address in the stack pointer, then post-increments SP by two.

Example:

```
MOVW      2, X+, 4, +Y
```

This example demonstrates how to work with data structures larger than bytes and words. With this instruction in a program loop, it is possible to move words of data from a list having one word per entry into a second table that has four bytes per table element. In this example the source pointer is updated after the data is read from memory (post-increment) while the destination pointer is updated before it is used to access memory (pre-increment).

### 3.8.6 Accumulator Offset Indexed Addressing

In this indexed addressing mode, the effective address is the sum of the values in the base index register and an unsigned offset in one of the accumulators. The value in the index register itself is not changed. The index register can be X, Y, SP, or PC and the accumulator can be either of the 8-bit accumulators (A or B) or the 16-bit D accumulator.

Example:

```
LDAA      B, X
```

This instruction internally adds B to X to form the address from which A will be loaded. B and X are not changed by this instruction. This example is similar to the following two-instruction combination in an M68HC11.

```
ABX
LDAA      0, X
```

However, this two-instruction sequence alters the index register. If this sequence was part of a loop where B changed on each pass, the index register would have to be re-loaded with the reference value on each loop pass. The use of LDAA B,X is more efficient in the CPU12.

### 3.8.7 Accumulator D Indirect Indexed Addressing

This indexed addressing mode adds the value in the D accumulator to the value in the base index register to form the address of a memory location that contains a pointer to the memory location affected by the instruction. The instruction operand does not point to the address of the memory location to be acted upon, but rather to the location of a pointer to the address to be acted upon. The square brackets distinguish this addressing mode from D accumulator offset indexing.

Example:

JMP	[D, PC]	
GO1	DC.W	PLACE1
GO2	DC.W	PLACE2
GO3	DC.W	PLACE3



This example is a computed GOTO. The values beginning at GO1 are addresses of potential destinations of the jump instruction. At the time the JMP [D,PC] instruction is executed, PC points to the address GO1, and D holds one of the values \$0000, \$0002, or \$0004 (determined by the program some time before the JMP).

Assume that the value in D is \$0002. The JMP instruction adds the values in D and PC to form the address of GO2. Next the CPU reads the address PLACE2 from memory at GO2 and jumps to PLACE2. The locations of PLACE1 through PLACE3 were known at the time of program assembly but the destination of the JMP depends upon the value in D computed during program execution.

### 3.9 Instructions Using Multiple Modes

Several CPU12 instructions use more than one addressing mode in the course of execution.

#### 3.9.1 Move Instructions

Move instructions use separate addressing modes to access the source and destination of a move. There are move variations for most combinations of immediate, extended, and indexed addressing modes.

The only combinations of addressing modes that are not allowed are those with an immediate mode destination (the operand of an immediate mode instruction is data, not an address). For indexed moves, the reference index register may be X, Y, SP, or PC.

Move instructions do not support indirect modes, or 9- or 16-bit offset modes requiring extra extension bytes. There are special considerations when using PC-relative addressing with move instructions.

PC-relative addressing uses the address of the location immediately following the last byte of object code for the current instruction as a reference point. The CPU12 normally corrects for queue offset and for instruction alignment so that queue operation is transparent to the user. However, move instructions pose three special problems:

1. Some moves use an indexed source and an indexed destination.
2. Some moves have object code that is too long to fit in the queue all at one time, so the PC value changes during execution.
3. All moves do not have the indexed postbyte as the last byte of object code.

These cases are not handled by automatic queue pointer maintenance, but it is still possible to use PC-relative indexing with move instructions by providing for PC offsets in source code.

**Table 3-3** shows PC offsets from the location immediately following the current instruction by addressing mode.

**Table 3-3 PC Offsets for Move Instructions**

MOVE Instruction	Addressing Modes	Offset Value
MOVB	IMM $\Rightarrow$ IDX	+ 1
	EXT $\Rightarrow$ IDX	+ 2
	IDX $\Rightarrow$ EXT	- 2
	IDX $\Rightarrow$ IDX	- 1 for 1st Operand + 1 for 2nd Operand
MOVW	IMM $\Rightarrow$ IDX	+ 2
	EXT $\Rightarrow$ IDX	+ 2
	IDX $\Rightarrow$ EXT	- 2
	IDX $\Rightarrow$ IDX	- 1 for 1st Operand + 1 for 2nd Operand

Example:

```
1000 18 09 C2 20 00    MOVB $2000 2,PC
```

Moves a byte of data from \$2000 to \$1009

The expected location of the PC = \$1005. The offset = +2.

$(1005 + 2 \text{ (for 2,PC)} + 2 \text{ (for correction)}) = 1009$

\$18 is the page pre-byte, 09 is the MOVB opcode for ext-idx, C2 is the indexed post-byte for 2,PC (without correction).

The Motorola MCUasm assembler produces corrected object code for PC-relative moves (18 09 C0 20 00 for the example shown). Note that, instead of assembling the 2,PC as C2, the correction has been applied to make it C0. Check whether an assembler makes the correction before using PC-relative moves.

### 3.9.2 Bit Manipulation Instructions

Bit manipulation instructions use either a combination of two or a combination of three addressing modes.

The BCLR and BSET instructions use an 8-bit mask to determine which bits in a memory byte are to be changed. The mask must be supplied with the instruction as an immediate mode value. The memory location to be modified can be specified by means of direct, extended, or indexed addressing modes.

The BRCLR and BRSET instructions use an 8-bit mask to test the states of bits in a memory byte. The mask is supplied with the instruction as an immediate mode value. The memory location to be tested is specified by means of direct, extended, or indexed addressing modes. Relative addressing mode is used to determine the branch address. A signed 8-bit offset must be supplied with the instruction.

### 3.10 Addressing More than 64 Kbytes

Some M68HC12 devices incorporate hardware that supports addressing a larger memory space than the standard 64 Kbytes. The expanded memory system uses fast on-chip logic to implement a transparent bank-switching scheme.

Increased code efficiency is the greatest advantage of using a switching scheme instead of a large linear address space. In systems with large linear address spaces, instructions require more bits of information to address a memory location, and CPU overhead is greater. Other advantages include the ability to change the size of system memory and the ability to use various types of external memory.

However, the add-on bank switching schemes used in other microcontrollers have known weaknesses. These include the cost of external glue logic, increased programming overhead to change banks, and the need to disable interrupts while banks are switched.

The M68HC12 system requires no external glue logic. Bank switching overhead is reduced by implementing control logic in the MCU. Interrupts do not need to be disabled during switching because switching tasks are incorporated in special instructions that greatly simplify program access to extended memory.

MCUs with expanded memory treat the 16 Kbytes of memory space from \$8000 to \$BFFF as a program memory window. Expanded-memory devices also have an 8-bit program page register (PPAGE), which allows up to 256 16-Kbyte program memory pages to be switched into and out of the program memory window. This provides for up to 4 Megabytes of paged program memory.

The CPU12 instruction set includes CALL and RTC (return from call) instructions, which greatly simplify the use of expanded memory space. These instructions also execute correctly on devices that do not have expanded-memory addressing capability, thus providing for portable code.

The CALL instruction is similar to the JSR instruction. When CALL is executed, the current value in PPAGE is pushed onto the stack with a return address, and a new instruction-supplied value is written to PPAGE. This value selects the page the called subroutine resides upon, and can be considered to be part of the effective address. For all addressing mode variations except indexed indirect modes, the new page value is provided by an immediate operand in the instruction. For indexed indirect variations of CALL, a pointer specifies memory locations where the new page value and the address of the called subroutine are stored. Use of indirect addressing for both the page value and the address within the page frees the program from keeping track of explicit values for either address.

The RTC instruction restores the saved program page value and the return address from the stack. This causes execution to resume at the next instruction after the original CALL instruction.

Refer to **SECTION 10 MEMORY EXPANSION** for a detailed discussion of memory expansion.

## SECTION 4 INSTRUCTION QUEUE

The CPU12 uses an instruction queue to increase execution speed. This section describes queue operation during normal program execution and changes in execution flow. These concepts augment the descriptions of instructions and cycle-by-cycle instruction execution in subsequent sections, but it is important to note that queue operation is automatic, and generally transparent to the user.

The material in this section is general. **SECTION 6 INSTRUCTION GLOSSARY** contains detailed information concerning cycle-by-cycle execution of each instruction. **SECTION 8 DEVELOPMENT AND DEBUG SUPPORT** contains detailed information about tracking queue operation and instruction execution.

### 4.1 Queue Description

The fetching mechanism in the CPU12 is best described as a queue rather than as a pipeline. Queue logic fetches program information and positions it for execution, but instructions are executed sequentially. A typical pipelined CPU can execute more than one instruction at the same time, but interactions between the prefetch and execution mechanisms can make tracking and debugging difficult. The CPU12 thus gains the advantages of independent fetches, yet maintains a straightforward relationship between bus and execution cycles.

There are two 16-bit queue stages and one 16-bit buffer. Program information is fetched in aligned 16-bit words. Unless buffering is required, program information is first queued into stage 1, then advanced to stage 2 for execution.

At least two words of program information are available to the CPU when execution begins. The first byte of object code is in either the even or odd half of the word in stage 2, and at least two more bytes of object code are in the queue.

Queue logic manages the position of program information so that the CPU itself does not deal with alignment. As it is executed, each instruction initiates at least enough program word fetches to replace its own object code in the queue.

The buffer is used when a program word arrives before the queue can advance. This occurs during execution of single-byte and odd-aligned instructions. For instance, the queue cannot advance after an aligned, single-byte instruction is executed, because the first byte of the next instruction is also in stage 2. In these cases, information is latched into the buffer until the queue can advance.

Two external pins, IPIPE[1:0], provide time-multiplexed information about data movement in the queue and instruction execution. Decoding and use of these signals is discussed in **SECTION 8 DEVELOPMENT AND DEBUG SUPPORT**.

## **4.2 Data Movement in the Queue**

All queue operations are combinations of four basic queue movement cycles. Descriptions of each of these cycles follows. Queue movement cycles are only one factor in instruction execution time, and should not be confused with bus cycles.

### **4.2.1 No Movement**

There is no data movement in the instruction queue during the cycle. This occurs during execution of instructions that must perform a number of internal operations, such as division instructions.

### **4.2.2 Latch Data from Bus**

All instructions initiate fetches to refill the queue as execution proceeds. However, a number of conditions, including instruction alignment and the length of previous instructions, affect when the queue advances. If the queue is not ready to advance when fetched information arrives, the information is latched into the buffer. Later, when the queue does advance, stage 1 is refilled from the buffer. If more than one latch cycle occurs before the queue advances, the buffer is filled on the first latch event and subsequent latch events are ignored until the queue advances.

### **4.2.3 Advance and Load from Data Bus**

The content of queue stage 1 advances to stage 2, and stage 1 is loaded with a word of program information from the data bus. The information was requested two bus cycles earlier but has only become available this cycle, due to access delay.

### **4.2.4 Advance and Load from Buffer**

The content of queue stage 1 advances to stage 2, and stage 1 is loaded with a word of program information from the buffer. The information in the buffer was latched from the data bus during a previous cycle because the queue was not ready to advance when it arrived.

## **4.3 Changes in Execution Flow**

During normal instruction execution, queue operations proceed as a continuous sequence of queue movement cycles. However, situations arise which call for changes in flow. These changes are categorized as resets, interrupts, subroutine calls, conditional branches, and jumps. Generally speaking, resets and interrupts are considered to be related to events outside the current program context that require special processing, while subroutine calls, branches, and jumps are considered to be elements of program structure.

During design, great care is taken to assure that the mechanism that increases instruction throughput during normal program execution does not cause bottlenecks during changes of program flow, but internal queue operation is largely transparent to the user. The following information is provided to enhance subsequent descriptions of instruction execution.

#### 4.3.1 Exceptions

Exceptions are events that require processing outside the normal flow of instruction execution. CPU12 exceptions include four types of resets, an unimplemented opcode trap, a software interrupt instruction, X-bit interrupts, and I-bit interrupts. All exceptions use the same microcode, but the CPU follows different execution paths for each type of exception.

CPU12 exception handling is designed to minimize the effect of queue operation on context switching. Thus, an exception vector fetch is the first part of exception processing, and fetches to refill the queue from the address pointed to by the vector are interleaved with the stacking operations that preserve context, so that program access time does not delay the switch. Refer to **SECTION 7 EXCEPTION PROCESSING** for detailed information.

#### 4.3.2 Subroutines

The CPU12 can branch to (BSR), jump to (JSR), or CALL subroutines. BSR and JSR are used to access subroutines in the normal 64-Kbyte address space. The CALL instruction is intended for use in MCUs with expanded memory capability.

BSR uses relative addressing mode to generate the effective address of the subroutine, while JSR can use various other addressing modes. Both instructions calculate a return address, stack the address, then perform three program word fetches to refill the queue. The first two words fetched are queued during the second and third cycles of the sequence. The third fetch cycle is performed in anticipation of a queue advance, which may occur during the fourth cycle of the sequence. If the queue is not yet ready to advance at that time, the third word of program information is held in the buffer.

Subroutines in the normal 64-Kbyte address space are terminated with a return from subroutine (RTS) instruction. RTS unstacks the return address, then performs three program word fetches from that address to refill the queue.

CALL is similar to JSR. MCUs with expanded memory treat 16 Kbytes of addresses from \$8000 to \$BFFF as a memory window. An 8-bit PPAGE register switches memory pages into and out of the window. When CALL is executed, a return address is calculated, then it and the current PPAGE value are stacked, and a new instruction-supplied value is written to PPAGE. The subroutine address is calculated, then three program word fetches are made from that address.

The RTC instruction is used to terminate subroutines in expanded memory. RTC unstacks the PPAGE value and the return address, then performs three program word fetches from that address to refill the queue.

CALL and RTC execute correctly in the normal 64-Kbyte address space, thus providing for portable code. However, since extra execution cycles are required, routinely substituting CALL/RTC for JSR/RTS is not recommended.

### 4.3.3 Branches

Branch instructions cause execution flow to change when specific pre-conditions exist. The CPU12 instruction set includes short conditional branches, long conditional branches, and bit-condition branches. Types and conditions of branch instructions are described in **5.18 Branch Instructions**. All branch instructions affect the queue similarly, but there are differences in overall cycle counts between the various types. Loop primitive instructions are a special type of branch instruction used to implement counter-based loops.

Branch instructions have two execution cases. Either the branch condition is satisfied, and a change of flow takes place, or the condition is not satisfied, and no change of flow occurs.

#### 4.3.3.1 Short Branches

The “not-taken” case for short branches is simple. Since the instruction consists of a single word containing both an opcode and an 8-bit offset, the queue advances, another program word is fetched, and execution continues with the next instruction.

The “taken” case for short branches requires that the queue be refilled so that execution can continue at a new address. First, the effective address of the destination is calculated using the relative offset in the instruction. Then, the address is loaded into the program counter, and the CPU performs three program word fetches at the new address. The first two words fetched are loaded into the instruction queue during the second and third cycles of the sequence. The third fetch cycle is performed in anticipation of a queue advance, which may occur during the first cycle of the next instruction. If the queue is not yet ready to advance at that time, the third word of program information is held in the buffer.

#### 4.3.3.2 Long Branches

The “not-taken” case for all long branches requires three cycles, while the “taken” case requires four cycles. This is due to differences in the amount of program information needed to fill the queue.

Long branch instructions begin with a \$18 prebyte which indicates that the opcode is on page 2 of the opcode map. The CPU12 treats the prebyte as a special one-byte instruction. If the prebyte is not aligned, the first cycle is used to perform a program word access; if the prebyte is aligned, the first cycle is used to perform a free cycle. The first cycle for the prebyte is executed whether or not the branch is taken.

The first cycle of the branch instruction is an optional cycle. Optional cycles make the effects of byte-sized and misaligned instructions consistent with those of aligned word-length instructions. Optional cycles are always performed, but serve different purposes determined by instruction alignment. Program information is always fetched as aligned 16-bit words. When an instruction consists of an odd number of bytes, and the first byte is aligned with an even byte boundary, an optional cycle is used to make an additional program word access that maintains queue order. In all other cases, the optional cycle appears as a free cycle.



In the "not-taken" case, the queue must advance so that execution can continue with the next instruction. Two cycles are used to refill the queue. Alignment determines how the second of these cycles is used.

In the "taken" case, the effective address of the branch is calculated using the 16-bit relative offset contained in the second word of the instruction. This address is loaded into the program counter, then the CPU performs three program word fetches at the new address. The first two words fetched are loaded into the instruction queue during the second and third cycles of the sequence. The third fetch cycle is performed in anticipation of a queue advance, which may occur during the first cycle of the next instruction. If the queue is not yet ready to advance, the third word of program information is held in the buffer.

#### **4.3.3.3 Bit Condition Branches**

Bit-conditional branch instructions read a location in memory, and branch if the bits in that location are in a certain state. These instructions can use direct, extended, or indexed addressing modes. Indexed operations require varying amounts of information to determine the effective address, so instruction length varies according to the mode used, which in turn affects the amount of program information fetched. In order to shorten execution time, these branches perform one program word fetch in anticipation of the "taken" case. The data from this fetch is overwritten by subsequent fetches in the "not-taken" case.

#### **4.3.3.4 Loop Primitives**

The loop primitive instructions test a counter value in a register or accumulator, and branch to an address specified by a 9-bit relative offset contained in the instruction if a specified pre-condition is met. There are auto-increment and auto-decrement versions of the instructions. The test and increment/decrement operations are performed on internal CPU registers, and require no additional program information. In order to shorten execution time, these branches perform one program word fetch in anticipation of the "taken" case. The data from this fetch is overwritten by subsequent fetches in the "not-taken" case. The "taken" case performs two additional program word fetches at the new address. In the "not-taken" case, the queue must advance so that execution can continue with the next instruction. Two cycles are used to refill the queue.

#### **4.3.4 Jumps**

JMP is the simplest change of flow instruction. JMP can use extended or indexed addressing. Indexed operations require varying amounts of information to determine the effective address, so instruction length varies according to the mode used, which in turn affects the amount of program information fetched. All forms of JMP perform three program word fetches at the new address. The first two words fetched are loaded into the instruction queue during the second and third cycles of the sequence. The third fetch cycle is performed in anticipation of a queue advance, which may occur during the first cycle of the next instruction. If the queue is not yet ready to advance, the third word of program information is held in the buffer.





## SECTION 5 INSTRUCTION SET OVERVIEW

This section contains general information about the CPU12 instruction set. It is organized into instruction categories grouped by function.

### 5.1 Instruction Set Description

CPU12 instructions are a superset of the M68HC11 instruction set. Code written for an M68HC11 can be reassembled and run on a CPU12 with no changes. The CPU12 provides expanded functionality and increased code efficiency.

In the M68HC12 architecture, all memory and I/O are mapped in a common 64-Kbyte address space (memory-mapped I/O). This allows the same set of instructions to be used to access memory, I/O, and control registers. General-purpose load, store, transfer, exchange, and move instructions facilitate movement of data to and from memory and peripherals.

The CPU12 has a full set of 8-bit and 16-bit mathematical instructions. There are instructions for signed and unsigned arithmetic, division and multiplication with 8-bit, 16-bit, and some larger operands.

Special arithmetic and logic instructions aid stacking operations, indexing, BCD calculation, and condition code register manipulation. There are also dedicated instructions for multiply and accumulate operations, table interpolation, and specialized fuzzy logic operations that involve mathematical calculations.

Refer to **SECTION 6 INSTRUCTION GLOSSARY** for detailed information about individual instructions. **APPENDIX A INSTRUCTION REFERENCE** contains quick-reference material, including an opcode map and postbyte encoding for indexed addressing, transfer/exchange instructions, and loop primitive instructions.

### 5.2 Load and Store Instructions

Load instructions copy memory content into an accumulator or register. Memory content is not changed by the operation. Load instructions (but not LEA\_ instructions) affect condition code bits so no separate test instructions are needed to check the loaded values for negative or zero conditions.

Store instructions copy the content of a CPU register to memory. Register/accumulator content is not changed by the operation. Store instructions automatically update the N and Z condition code bits, which can eliminate the need for a separate test instruction in some programs.

**Table 5-1** is a summary of load and store instructions.

**Table 5-1 Load and Store Instructions**

Load Instructions		
Mnemonic	Function	Operation
LDAA	Load A	$(M) \Rightarrow A$
LDAB	Load B	$(M) \Rightarrow B$
LDD	Load D	$(M : M + 1) \Rightarrow (A:B)$
LDS	Load SP	$(M : M + 1) \Rightarrow SP$
LDX	Load Index Register X	$(M : M + 1) \Rightarrow X$
LDY	Load Index Register Y	$(M : M + 1) \Rightarrow Y$
LEAS	Load Effective Address into SP	Effective Address $\Rightarrow$ SP
LEAX	Load Effective Address into X	Effective Address $\Rightarrow$ X
LEAY	Load Effective Address into Y	Effective Address $\Rightarrow$ Y
Store Instructions		
Mnemonic	Function	Operation
STAA	Store A	$(A) \Rightarrow M$
STAB	Store B	$(B) \Rightarrow M$
STD	Store D	$(A) \Rightarrow M, (B) \Rightarrow M + 1$
STS	Store SP	$(SP) \Rightarrow M : M + 1$
STX	Store X	$(X) \Rightarrow M : M + 1$
STY	Store Y	$(Y) \Rightarrow M : M + 1$

### 5.3 Transfer and Exchange Instructions

Transfer instructions copy the content of a register or accumulator into another register or accumulator. Source content is not changed by the operation. TFR is a universal transfer instruction, but other mnemonics are accepted for compatibility with the M68HC11. The TAB and TBA instructions affect the N, Z, and V condition code bits in the same way as M68HC11 instructions. The TFR instruction does not affect the condition code bits.

Exchange instructions exchange the contents of pairs of registers or accumulators.

The SEX instruction is a special case of the universal transfer instruction that is used to sign-extend 8-bit two's complement numbers so that they can be used in 16-bit operations. The 8-bit number is copied from accumulator A, accumulator B, or the condition codes register to accumulator D, the X index register, the Y index register, or the stack pointer. All the bits in the upper byte of the 16-bit result are given the value of the MSB of the 8-bit number.

**SECTION 6 INSTRUCTION GLOSSARY** contains information concerning other transfers and exchanges between 8- and 16-bit registers.

**Table 5-2** is a summary of transfer and exchange instructions.

**Table 5-2 Transfer and Exchange Instructions**

Transfer Instructions		
Mnemonic	Function	Operation
TAB	Transfer A to B	$(A) \Rightarrow B$
TAP	Transfer A to CCR	$(A) \Rightarrow \text{CCR}$
TBA	Transfer B to A	$(B) \Rightarrow A$
TFR	Transfer Register to Register	$(A, B, \text{CCR}, D, X, Y, \text{ or } SP) \Rightarrow A, B, \text{CCR}, D, X, Y, \text{ or } SP$
TPA	Transfer CCR to A	$(\text{CCR}) \Rightarrow A$
TSX	Transfer SP to X	$(SP) \Rightarrow X$
TSY	Transfer SP to Y	$(SP) \Rightarrow Y$
TXS	Transfer X to SP	$(X) \Rightarrow SP$
TYS	Transfer Y to SP	$(Y) \Rightarrow SP$
Exchange Instructions		
Mnemonic	Function	Operation
EXG	Exchange Register to Register	$(A, B, \text{CCR}, D, X, Y, \text{ or } SP) \Leftrightarrow (A, B, \text{CCR}, D, X, Y, \text{ or } SP)$
XGDX	Exchange D with X	$(D) \Leftrightarrow (X)$
XGDY	Exchange D with Y	$(D) \Leftrightarrow (Y)$
Sign Extension Instruction		
Mnemonic	Function	Operation
SEX	Sign Extend 8-Bit Operand	$(A, B, \text{CCR}) \Rightarrow X, Y, \text{ or } SP$

#### 5.4 Move Instructions

These instructions move data bytes or words from a source ( $M_1, M : M + 1_1$ ) to a destination ( $M_2, M : M + 1_2$ ) in memory. Six combinations of immediate, extended, and indexed addressing are allowed to specify source and destination addresses ( $\text{IMM} \Rightarrow \text{EXT}, \text{IMM} \Rightarrow \text{IDX}, \text{EXT} \Rightarrow \text{EXT}, \text{EXT} \Rightarrow \text{IDX}, \text{IDX} \Rightarrow \text{EXT}, \text{IDX} \Rightarrow \text{IDX}$ ).

Table 5-3 shows byte and word move instructions.

**Table 5-3 Move Instructions**

Mnemonic	Function	Operation
MOVB	Move Byte (8-bit)	$(M_1) \Rightarrow M_2$
MOVW	Move Word (16-bit)	$(M : M + 1_1) \Rightarrow M : M + 1_2$

#### 5.5 Addition and Subtraction Instructions

Signed and unsigned 8- and 16-bit addition can be performed between registers or between registers and memory. Special instructions support index calculation. Instructions that add the CCR carry bit facilitate multiple precision computation.

Signed and unsigned 8- and 16-bit subtraction can be performed between registers or between registers and memory. Special instructions support index calculation. Instructions that subtract the CCR carry bit facilitate multiple precision computation. Refer to Table 5-4 for addition and subtraction instructions.

**Table 5-4 Addition and Subtraction Instructions**

Addition Instructions		
Mnemonic	Function	Operation
ABA	Add A to B	$(A) + (B) \Rightarrow A$
ABX	Add B to X	$(B) + (X) \Rightarrow X$
ABY	Add B to Y	$(B) + (Y) \Rightarrow Y$
ADCA	Add with Carry to A	$(A) + (M) + C \Rightarrow A$
ADCB	Add with Carry to B	$(B) + (M) + C \Rightarrow B$
ADDA	Add without Carry to A	$(A) + (M) \Rightarrow A$
ADDB	Add without Carry to B	$(B) + (M) \Rightarrow B$
ADDD	Add to D	$(A:B) + (M : M + 1) \Rightarrow A : B$
Subtraction Instructions		
Mnemonic	Function	Operation
SBA	Subtract B from A	$(A) - (B) \Rightarrow A$
SBCA	Subtract with Borrow from A	$(A) - (M) - C \Rightarrow A$
SBCB	Subtract with Borrow from B	$(B) - (M) - C \Rightarrow B$
SUBA	Subtract Memory from A	$(A) - (M) \Rightarrow A$
SUBB	Subtract Memory from B	$(B) - (M) \Rightarrow B$
SUBD	Subtract Memory from D (A:B)	$(D) - (M : M + 1) \Rightarrow D$

## 5.6 Binary Coded Decimal Instructions

To add binary coded decimal operands, use addition instructions that set the half-carry bit in the CCR, then adjust the result with the DAA instruction. **Table 5-5** is a summary of instructions that can be used to perform BCD operations.

**Table 5-5 BCD Instructions**

Mnemonic	Function	Operation
ABA	Add B to A	$(A) + (B) \Rightarrow A$
ADCA	Add with Carry to A	$(A) + (M) + C \Rightarrow A$
ADCB	Add with Carry to B	$(B) + (M) + C \Rightarrow B$
ADDA	Add Memory to A	$(A) + (M) \Rightarrow A$
ADDB	Add Memory to B	$(B) + (M) \Rightarrow B$
DAA	Decimal Adjust A	$(A)_{10}$

## 5.7 Decrement and Increment Instructions

These instructions are optimized 8- and 16-bit addition and subtraction operations. They are generally used to implement counters. Because they do not affect the carry bit in the CCR, they are particularly well suited for loop counters in multiple-precision computation routines. Refer to **5.19 Loop Primitive Instructions** for information concerning automatic counter branches. **Table 5-6** is a summary of decrement and increment instructions.

**Table 5-6 Decrement and Increment Instructions**

Decrement Instructions		
Mnemonic	Function	Operation
DEC	Decrement Memory	$(M) - \$01 \Rightarrow M$
DECA	Decrement A	$(A) - \$01 \Rightarrow A$
DECB	Decrement B	$(B) - \$01 \Rightarrow B$
DES	Decrement SP	$(SP) - \$0001 \Rightarrow SP$
DEX	Decrement X	$(X) - \$0001 \Rightarrow X$
DEY	Decrement Y	$(Y) - \$0001 \Rightarrow Y$
Increment Instructions		
Mnemonic	Function	Operation
INC	Increment Memory	$(M) + \$01 \Rightarrow M$
INCA	Increment A	$(A) + \$01 \Rightarrow A$
INCB	Increment B	$(B) + \$01 \Rightarrow B$
INS	Increment SP	$(SP) + \$0001 \Rightarrow SP$
INX	Increment X	$(X) + \$0001 \Rightarrow X$
INY	Increment Y	$(Y) + \$0001 \Rightarrow Y$

## 5.8 Compare and Test Instructions

Compare and test instructions perform subtraction between a pair of registers or between a register and memory. The result is not stored, but condition codes are set by the operation. These instructions are generally used to establish conditions for branch instructions. In this architecture, most instructions update condition code bits automatically, so it is often unnecessary to include separate test or compare instructions. **Table 5-7** is a summary of compare and test instructions.

**Table 5-7 Compare and Test Instructions**

Compare Instructions		
Mnemonic	Function	Operation
CBA	Compare A to B	$(A) - (B)$
CMPA	Compare A to Memory	$(A) - (M)$
CMPB	Compare B to Memory	$(B) - (M)$
CPD	Compare D to Memory (16-bit)	$(A : B) - (M : M + 1)$
CPS	Compare SP to Memory (16-bit)	$(SP) - (M : M + 1)$
CPX	Compare X to Memory (16-bit)	$(X) - (M : M + 1)$
CPY	Compare Y to Memory (16-bit)	$(Y) - (M : M + 1)$
Test Instructions		
Mnemonic	Function	Operation
TST	Test Memory for Zero or Minus	$(M) - \$00$
TSTA	Test A for Zero or Minus	$(A) - \$00$
TSTB	Test B for Zero or Minus	$(B) - \$00$

## 5.9 Boolean Logic Instructions

These instructions perform a logic operation between an 8-bit accumulator or the CCR and a memory value. AND, OR, and exclusive OR functions are supported. **Table 5-8** summarizes logic instructions.

**Table 5-8 Boolean Logic Instructions**

Mnemonic	Function	Operation
ANDA	AND A with Memory	$(A) \bullet (M) \Rightarrow A$
ANDB	AND B with Memory	$(B) \bullet (M) \Rightarrow B$
ANDCC	AND CCR with Memory (Clear CCR Bits)	$(CCR) \bullet (M) \Rightarrow CCR$
EORA	Exclusive OR A with Memory	$(A) \oplus (M) \Rightarrow A$
EORB	Exclusive OR B with Memory	$(B) \oplus (M) \Rightarrow B$
ORAA	OR A with Memory	$(A) + (M) \Rightarrow A$
ORAB	OR B with Memory	$(B) + (M) \Rightarrow B$
ORCC	OR CCR with Memory (Set CCR Bits)	$(CCR) + (M) \Rightarrow CCR$

## 5.10 Clear, Complement, and Negate Instructions

Each of these instructions performs a specific binary operation on a value in an accumulator or in memory. Clear operations clear the value to zero, complement operations replace the value with its one's complement, and negate operations replace the value with its two's complement. **Table 5-9** is a summary of clear, complement and negate instructions.

**Table 5-9 Clear, Complement, and Negate Instructions**

Mnemonic	Function	Operation
CLC	Clear C Bit in CCR	$0 \Rightarrow C$
CLI	Clear I Bit in CCR	$0 \Rightarrow I$
CLR	Clear Memory	$\$00 \Rightarrow M$
CLRA	Clear A	$\$00 \Rightarrow A$
CLRB	Clear B	$\$00 \Rightarrow B$
CLV	Clear V bit in CCR	$0 \Rightarrow V$
COM	One's Complement Memory	$\$FF - (M) \Rightarrow M$ or $(\bar{M}) \Rightarrow M$
COMA	One's Complement A	$\$FF - (A) \Rightarrow A$ or $(\bar{A}) \Rightarrow A$
COMB	One's Complement B	$\$FF - (B) \Rightarrow B$ or $(\bar{B}) \Rightarrow B$
NEG	Two's Complement Memory	$\$00 - (M) \Rightarrow M$ or $(\bar{M}) + 1 \Rightarrow M$
NEGA	Two's Complement A	$\$00 - (A) \Rightarrow A$ or $(\bar{A}) + 1 \Rightarrow A$
NEGB	Two's Complement B	$\$00 - (B) \Rightarrow B$ or $(\bar{B}) + 1 \Rightarrow B$

### 5.11 Multiplication and Division Instructions

There are instructions for signed and unsigned 8- and 16-bit multiplication. Eight-bit multiplication operations have a 16-bit product. Sixteen-bit multiplication operations have 32-bit products.

Integer and fractional division instructions have 16-bit dividend, divisor, quotient, and remainder. Extended division instructions use a 32-bit dividend and a 16-bit divisor to produce a 16-bit quotient and a 16-bit remainder.

**Table 5-10** is a summary of multiplication and division instructions.

**Table 5-10 Multiplication and Division Instructions**

Multiplication Instructions		
Mnemonic	Function	Operation
EMUL	16 by 16 Multiply (Unsigned)	$(D) \times (Y) \Rightarrow Y : D$
EMULS	16 by 16 Multiply (Signed)	$(D) \times (Y) \Rightarrow Y : D$
MUL	8 by 8 Multiply (Unsigned)	$(A) \times (B) \Rightarrow A : B$
Division Instructions		
Mnemonic	Function	Operation
EDIV	32 by 16 Divide (Unsigned)	$(Y : D) \div (X)$ Quotient $\Rightarrow Y$ Remainder $\Rightarrow D$
EDIVS	32 by 16 Divide (Signed)	$(Y : D) \div (X)$ Quotient $\Rightarrow Y$ Remainder $\Rightarrow D$
FDIV	16 by 16 Fractional Divide	$(D) \div (X) \Rightarrow X$ remainder $\Rightarrow D$
IDIV	16 by 16 Integer Divide (Unsigned)	$(D) \div (X) \Rightarrow X$ remainder $\Rightarrow D$
IDIVS	16 by 16 Integer Divide (Signed)	$(D) \div (X) \Rightarrow X$ remainder $\Rightarrow D$

### 5.12 Bit Test and Manipulation Instructions

These operations use a mask value to test or change the value of individual bits in an accumulator or in memory. BITA and BITB provide a convenient means of testing bits without altering the value of either operand. **Table 5-11** is a summary of bit test and manipulation instructions.

**Table 5-11 Bit Test and Manipulation Instructions**

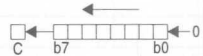
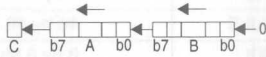
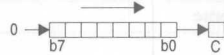
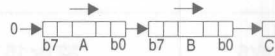
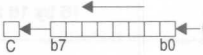
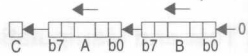
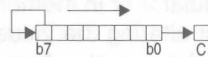
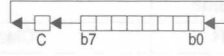
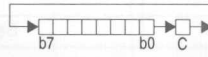
Mnemonic	Function	Operation
BCLR	Clear Bits in Memory	$(M) \bullet (\overline{mm}) \Rightarrow M$
BITA	Bit Test A	$(A) \bullet (M)$
BITB	Bit Test B	$(B) \bullet (M)$
BSET	Set Bits in Memory	$(M) + (mm) \Rightarrow M$



### 5.13 Shift and Rotate Instructions

There are shifts and rotates for all accumulators and for memory bytes. All pass the shifted-out bit through the C status bit to facilitate multiple-byte operations. Because logical and arithmetic left shifts are identical, there are no separate logical left shift operations. LSL mnemonics are assembled as ASL operations. **Table 5-12** shows shift and rotate instructions.

**Table 5-12 Shift and Rotate Instructions**

Logical Shifts		
Mnemonic	Function	Operation
LSL LSLA LSLB	Logic Shift Left Memory Logic Shift Left A Logic Shift Left B	
LSLD	Logic Shift Left D	
LSR LSRA LSRB	Logic Shift Right Memory Logic Shift Right A Logic Shift Right B	
LSRD	Logic Shift Right D	
Arithmetic Shifts		
Mnemonic	Function	Operation
ASL ASLA ASLB	Arithmetic Shift Left Memory Arithmetic Shift Left A Arithmetic Shift Left B	
ASLD	Arithmetic Shift Left D	
ASR ASRA ASRB	Arithmetic Shift Right Memory Arithmetic Shift Right A Arithmetic Shift Right B	
Rotates		
Mnemonic	Function	Operation
ROL ROLA ROLB	Rotate Left Memory Through Carry Rotate Left A Through Carry Rotate Left B Through Carry	
ROR RORA RORB	Rotate Right Memory Through Carry Rotate Right A Through Carry Rotate Right B Through Carry	

## 5.14 Fuzzy Logic Instructions

The CPU12 instruction set includes instructions that support efficient processing of fuzzy logic operations. The descriptions of fuzzy logic instructions that follow are functional overviews. **Table 5-13** summarizes the fuzzy logic instructions. Refer to **SECTION 9 FUZZY LOGIC SUPPORT** for detailed discussion.

### 5.14.1 Fuzzy Logic Membership Instruction

The MEM instruction is used during the fuzzification process. During fuzzification, current system input values are compared against stored input membership functions to determine the degree to which each label of each system input is true. This is accomplished by finding the y value for the current input on a trapezoidal membership function for each label of each system input. The MEM instruction performs this calculation for one label of one system input. To perform the complete fuzzification task for a system, several MEM instructions must be executed, usually in a program loop structure.

### 5.14.2 Fuzzy Logic Rule Evaluation Instructions

The REV and REVW instructions perform MIN-MAX rule evaluations that are central elements of a fuzzy logic inference program. Fuzzy input values are processed using a list of rules from the knowledge base to produce a list of fuzzy outputs. The REV instruction treats all rules as equally important. The REVW instruction allows each rule to have a separate weighting factor. The two rule evaluation instructions also differ in the way rules are encoded into the knowledge base. Because they require a number of cycles to execute, rule evaluation instructions can be interrupted. Once the interrupt has been serviced, instruction execution resumes at the point the interrupt occurred.

### 5.14.3 Fuzzy Logic Averaging Instruction

The WAV instruction provides a facility for weighted average calculations. In order to be usable, the fuzzy outputs produced by rule evaluation must be defuzzified to produce a single output value which represents the combined effect of all of the fuzzy outputs. Fuzzy outputs correspond to the labels of a system output and each is defined by a membership function in the knowledge base. The CPU12 typically uses singletons for output membership functions rather than the trapezoidal shapes used for inputs. As with inputs, the x-axis represents the range of possible values for a system output. Singleton membership functions consist of the x-axis position for a label of the system output. Fuzzy outputs correspond to the y-axis height of the corresponding output membership function. The WAV instruction calculates the numerator and denominator sums for a weighted average of the fuzzy outputs. Because WAV requires a number of cycles to execute, it can be interrupted. The wavr pseudo-instruction causes execution to resume at the point it was interrupted.

**Table 5-13 Fuzzy Logic Instructions**

Mnemonic	Function	Operation
MEM	Membership Function	$\mu(\text{grade}) \Rightarrow M(Y)$ $(X) + 4 \Rightarrow X; (Y) + 1 \Rightarrow Y; A \text{ unchanged}$ <p>if <math>(A) &lt; P1</math> or <math>(A) &gt; P2</math>, then <math>\mu = 0</math>, else  <math>\mu = \text{MIN} [((A) - P1) \times S1, (P2 - (A)) \times S2, \\$FF]</math>  where:  A = current crisp input value  X points to a four byte data structure that describes a trapezoidal membership function as base intercept points and slopes (P1, P2, S1, S2)  Y points at fuzzy input (RAM location)</p> <p>See instruction details for special cases</p>
REV	MIN-MAX Rule Evaluation	<p>Find smallest rule input (MIN)  Store to rule outputs unless fuzzy output is larger (MAX)</p> <p>Rules are unweighted</p> <p>Each rule input is an 8-bit offset from a base address in Y  Each rule output is an 8-bit offset from a base address in Y  \$FE separates rule inputs from rule outputs  \$FF terminates the rule list</p> <p>REV can be interrupted</p>
REWV	MIN-MAX Rule Evaluation	<p>Find smallest rule input (MIN)  Multiply by a rule weighting factor (optional)  Store to rule outputs unless fuzzy output is larger (MAX)</p> <p>Each rule input is the 16-bit address of a fuzzy input  Each rule output is the 16-bit address of a fuzzy output  Address \$FFFE separates rule inputs from rule outputs  \$FFFF terminates the rule list  Weights are 8-bit values in a separate table</p> <p>REWV can be interrupted</p>
WAV	Calculates Numerator (Sum of Products) and Denominator (Sum of Weights) for Weighted Average Calculation Results Are Placed in Correct Registers For EDIV Immediately After WAV	$\sum_{i=1}^B S_i F_i \Rightarrow Y:D$ $\sum_{i=1}^B F_i \Rightarrow X$
wavr	Resumes Execution of Interrupted WAV Instruction	Recover immediate results from stack rather than initializing them to zero.

### 5.15 Maximum and Minimum Instructions

These instructions are used to make comparisons between an accumulator and a memory location. These instructions can be used for linear programming operations, such as Simplex-method optimization or for fuzzification.

MAX and MIN instructions use accumulator A to perform 8-bit comparisons, while EMAX and EMIN instructions use accumulator D to perform 16-bit comparisons. The result (maximum or minimum value) can be stored in the accumulator (EMAXD, EMIND, MAXA, MINA) or the memory address (EMAXM, EMINM, MAXM, MINM).

Table 5-14 is a summary of minimum and maximum instructions.

**Table 5-14 Minimum and Maximum Instructions**

Minimum Instructions		
Mnemonic	Function	Operation
EMIND	MIN of Two Unsigned 16-Bit Values Result to Accumulator	$\text{MIN}((D), (M : M + 1)) \Rightarrow D$
EMINM	MIN of Two Unsigned 16-Bit Values Result to Memory	$\text{MIN}((D), (M : M + 1)) \Rightarrow M : M + 1$
MINA	MIN of Two Unsigned 8-Bit Values Result to Accumulator	$\text{MIN}((A), (M)) \Rightarrow A$
MINM	MIN of Two Unsigned 8-Bit Values Result to Memory	$\text{MIN}((A), (M)) \Rightarrow M$
Maximum Instructions		
Mnemonic	Function	Operation
EMAXD	MAX of Two Unsigned 16-Bit Values Result to Accumulator	$\text{MAX}((D), (M : M + 1)) \Rightarrow D$
EMAXM	MAX of Two Unsigned 16-Bit Values Result to Memory	$\text{MAX}((D), (M : M + 1)) \Rightarrow M : M + 1$
MAXA	MAX of Two Unsigned 8-Bit Values Result to Accumulator	$\text{MAX}((A), (M)) \Rightarrow A$
MAXM	MAX of Two Unsigned 8-Bit Values Result to Memory	$\text{MAX}((A), (M)) \Rightarrow M$

### 5.16 Multiply and Accumulate Instruction

The EMACS instruction multiplies two 16-bit operands stored in memory and accumulates the 32-bit result in a third memory location. EMACS can be used to implement simple digital filters and defuzzification routines that use 16-bit operands. The WAV instruction incorporates an 8- to 16-bit multiply and accumulate operation that obtains a numerator for the weighted average calculation. The EMACS instruction can automate this portion of the averaging operation when 16-bit operands are used. Table 5-15 shows the EMACS instruction.

**Table 5-15 Multiply and Accumulate Instructions**

Mnemonic	Function	Operation
EMACS	Multiply and Accumulate (Signed) 16 × 16 Bit ⇒ 32 Bit	$((M_{(X)}:M_{(X+1)}) \times (M_{(Y)}:M_{(Y+1)})) + (M \sim M + 3) \Rightarrow M \sim M + 3$

### 5.17 Table Interpolation Instructions

The TBL and ETBL instructions interpolate values from tables stored in memory. Any function that can be represented as a series of linear equations can be represented by a table of appropriate size. Interpolation can be used for many purposes, including tabular fuzzy logic membership functions. TBL uses 8-bit table entries and returns an 8-bit result; ETBL uses 16-bit table entries and returns a 16-bit result. Use of indexed addressing mode provides great flexibility in structuring tables.

Consider each of the successive values stored in a table to be y-values for the end-point of a line segment. The value in the B accumulator before instruction execution begins represents change in x from the beginning of the line segment to the lookup point divided by total change in x from the beginning to the end of the line segment. B is treated as an 8-bit binary fraction with radix point left of the MSB, so each line segment is effectively divided into 256 smaller segments. During instruction execution, the change in y between the beginning and end of the segment (a signed byte for TBL or a signed word for ETBL) is multiplied by the content of the B accumulator to obtain an intermediate delta-y term. The result (stored in the A accumulator by TBL, and in the D accumulator by ETBL) is the y-value of the beginning point plus the signed intermediate delta-y value. **Table 5-16** shows the table interpolation instructions.

**Table 5-16 Table Interpolation Instructions**

Mnemonic	Function	Operation
ETBL	16-Bit Table Lookup and Interpolate (no indirect addressing modes allowed)	$(M : M + 1) + [(B) \times ((M + 2 : M + 3) - (M : M + 1))] \Rightarrow D$ Initialize B, and index before ETBL. <ea> points to the first table entry (M : M + 1) B is fractional part of lookup value
TBL	8-Bit Table Lookup and Interpolate (no indirect addressing modes allowed.)	$(M) + [(B) \times ((M + 1) - (M))] \Rightarrow A$ Initialize B, and index before TBL. <ea> points to the first 8-bit table entry (M) B is fractional part of lookup value.

## 5.18 Branch Instructions

Branch instructions cause sequence to change when specific conditions exist. The CPU12 uses three kinds of branch instructions. These are short branches, long branches, and bit-conditional branches.

Branch instructions can also be classified by the type of condition that must be satisfied in order for a branch to be taken. Some instructions belong to more than one classification.

Unary branch instructions always execute.

Simple branches are taken when a specific bit in the condition code register is in a specific state as a result of a previous operation.

Unsigned branches are taken when comparison or test of unsigned quantities results in a specific combination of condition code register bits.

Signed branches are taken when comparison or test of signed quantities results in a specific combination of condition code register bits.

### 5.18.1 Short Branch Instructions

Short branch instructions operate as follows. When a specified condition is met, a signed 8-bit offset is added to the value in the program counter. Program execution continues at the new address.

The numeric range of short branch offset values is \$80 (–128) to \$7F (127) from the address of the next memory location after the offset value.

**Table 5-17** is a summary of the short branch instructions.

### 5.18.2 Long Branch Instructions

Long branch instructions operate as follows. When a specified condition is met, a signed 16-bit offset is added to the value in the program counter. Program execution continues at the new address. Long branches are used when large displacements between decision-making steps are necessary.

The numeric range of long branch offset values is \$8000 (–32,768) to \$7FFF (32,767) from the address of the next memory location after the offset value. This permits branching from any location in the standard 64-Kbyte address map to any other location in the map.

**Table 5-18** is a summary of the long branch instructions.

**Table 5-17 Short Branch Instructions**

Unary Branches			
Mnemonic	Function	Equation or Operation	
BRA	Branch Always	$1 = 1$	
BRN	Branch Never	$1 = 0$	
Simple Branches			
Mnemonic	Function	Equation or Operation	
BCC	Branch if Carry Clear	$C = 0$	
BCS	Branch if Carry Set	$C = 1$	
BEQ	Branch if Equal	$Z = 1$	
BMI	Branch if Minus	$N = 1$	
BNE	Branch if Not Equal	$Z = 0$	
BPL	Branch if Plus	$N = 0$	
BVC	Branch if Overflow Clear	$V = 0$	
BVS	Branch if Overflow Set	$V = 1$	
Unsigned Branches			
Mnemonic	Function	Relation	Equation or Operation
BHI	Branch if Higher	$R > M$	$C + Z = 0$
BHS	Branch if Higher or Same	$R \geq M$	$C = 0$
BLO	Branch if Lower	$R < M$	$C = 1$
BLS	Branch if Lower or Same	$R \leq M$	$C + Z = 1$
Signed Branches			
Mnemonic	Function	Relation	Equation or Operation
BGE	Branch if Greater Than or Equal	$R \geq M$	$N \oplus V = 0$
BGT	Branch if Greater Than	$R > M$	$Z + (N \oplus V) = 0$
BLE	Branch if Less Than or Equal	$R \leq M$	$Z + (N \oplus V) = 1$
BLT	Branch if Less Than	$R < M$	$N \oplus V = 1$

**Table 5-18 Long Branch Instructions**

Unary Branches		
Mnemonic	Function	Equation or Operation
LBRA	Long Branch Always	$1 = 1$
LBRN	Long Branch Never	$1 = 0$
Simple Branches		
Mnemonic	Function	Equation or Operation
LBCC	Long Branch if Carry Clear	$C = 0$
LBCS	Long Branch if Carry Set	$C = 1$
LBEQ	Long Branch if Equal	$Z = 1$
LBMI	Long Branch if Minus	$N = 1$
LBNE	Long Branch if Not Equal	$Z = 0$
LBPL	Long Branch if Plus	$N = 0$
LBVC	Long Branch if Overflow Clear	$V = 0$
LBVS	Long Branch if Overflow Set	$V = 1$
Unsigned Branches		
Mnemonic	Function	Equation or Operation
LBHI	Long Branch if Higher	$C + Z = 0$
LBHS	Long Branch if Higher or Same	$C = 0$
LBLO	Long Branch if Lower	$Z = 1$
LBLS	Long Branch if Lower or Same	$C + Z = 1$
Signed Branches		
Mnemonic	Function	Equation or Operation
LBGE	Long Branch if Greater Than or Equal	$N \oplus V = 0$
LBGT	Long Branch if Greater Than	$Z + (N \oplus V) = 0$
LBLE	Long Branch if Less Than or Equal	$Z + (N \oplus V) = 1$
LBLT	Long Branch if Less Than	$N \oplus V = 1$



### 5.18.3 Bit Condition Branch Instructions

These branches are taken when bits in a memory byte are in a specific state. A mask operand is used to test the location. If all bits in that location that correspond to ones in the mask are set (BRSET) or cleared (BRCLR), the branch is taken.

The numeric range of 8-bit offset values is \$80 (-128) to \$7F (127) from the address of the next memory location after the offset value. **Table 5-19** is a summary of bit-condition branches.

**Table 5-19 Bit Condition Branch Instructions**

Mnemonic	Function	Equation or Operation
BRCLR	Branch if Selected Bits Clear	$(M) \bullet (mm) = 0$
BRSET	Branch if Selected Bits Set	$(\bar{M}) \bullet (mm) = 0$

### 5.19 Loop Primitive Instructions

The loop primitives can also be thought of as counter branches. The instructions test a counter value in a register or accumulator (A, B, D, X, Y, or SP) for zero or nonzero value as a branch condition. There are predecrement, preincrement and test-only versions of these instructions.

The numeric range of 8-bit offset values is \$80 (-128) to \$7F (127) from the address of the next memory location after the offset value. **Table 5-20** is a summary of loop primitive branches.

**Table 5-20 Loop Primitive Instructions**

Mnemonic	Function	Equation or Operation
DBEQ	Decrement counter and branch if = 0 (counter = A, B, D, X, Y, or SP)	$(\text{counter}) - 1 \Rightarrow \text{counter}$ If (counter) = 0, then branch else continue to next instruction
DBNE	Decrement counter and branch if $\neq$ 0 (counter = A, B, D, X, Y, or SP)	$(\text{counter}) - 1 \Rightarrow \text{counter}$ If (counter) not = 0, then branch else continue to next instruction
IBEQ	Increment counter and branch if = 0 (counter = A, B, D, X, Y, or SP)	$(\text{counter}) + 1 \Rightarrow \text{counter}$ If (counter) = 0, then branch else continue to next instruction
IBNE	Increment counter and branch if $\neq$ 0 (counter = A, B, D, X, Y, or SP)	$(\text{counter}) + 1 \Rightarrow \text{counter}$ If (counter) not = 0, then branch else continue to next instruction
TBEQ	Test counter and branch if = 0 (counter = A, B, D, X, Y, or SP)	If (counter) = 0, then branch else continue to next instruction
TBNE	Test counter and branch if $\neq$ 0 (counter = A, B, D, X, Y, or SP)	If (counter) not = 0, then branch else continue to next instruction

## 5.20 Jump and Subroutine Instructions

Jump instructions cause immediate changes in sequence. The JMP instruction loads the PC with an address in the 64-Kbyte memory map, and program execution continues at that address. The address can be provided as an absolute 16-bit address or determined by various forms of indexed addressing.

Subroutine instructions optimize the process of transferring control to a code segment that performs a particular task. A short branch (BSR), a jump (JSR), or an expanded-memory call (CALL) can be used to initiate subroutines. There is no LBSR instruction, but a PC-relative JSR performs the same function. A return address is stacked, then execution begins at the subroutine address. Subroutines in the normal 64-Kbyte address space are terminated with an RTS instruction. RTS unstacks the return address so that execution resumes with the instruction after BSR or JSR.

The CALL instruction is intended for use with expanded memory. CALL stacks the value in the PPAGE register and the return address, then writes a new value to PPAGE to select the memory page where the subroutine resides. The page value is an immediate operand in all addressing modes except indexed indirect modes; in these modes, an operand points to locations in memory where the new page value and subroutine address are stored. The RTC instruction is used to terminate subroutines in expanded memory. RTC unstacks the PPAGE value and the return address so that execution resumes with the next instruction after CALL. For software compatibility, CALL and RTC execute correctly on devices that do not have expanded addressing capability. **Table 5-21** summarizes the jump and subroutine instructions.

**Table 5-21 Jump and Subroutine Instructions**

Mnemonic	Function	Operation
BSR	Branch to Subroutine	$SP - 2 \Rightarrow SP$ $RTN_H : RTN_L \Rightarrow M(SP) : M(SP+1)$ Subroutine address $\Rightarrow PC$
CALL	Call Subroutine in Expanded Memory	$SP - 2 \Rightarrow SP$ $RTN_H : RTN_L \Rightarrow M(SP) : M(SP+1)$ $SP - 1 \Rightarrow SP$ $(PPAGE) \Rightarrow M(SP)$ Page $\Rightarrow PPAGE$ Subroutine address $\Rightarrow PC$
JMP	Jump	Subroutine Address $\Rightarrow PC$
JSR	Jump to Subroutine	$SP - 2 \Rightarrow SP$ $RTN_H : RTN_L \Rightarrow M(SP) : M(SP+1)$ Subroutine address $\Rightarrow PC$
RTC	Return from Call	$M(SP) : M(SP+1) \Rightarrow PC_H : PC_L$ $SP + 2 \Rightarrow SP$
RTS	Return from Subroutine	$M(SP) \Rightarrow PPAGE$ $SP + 1 \Rightarrow SP$ $M(SP) : M(SP+1) \Rightarrow PC_H : PC_L$ $SP + 2 \Rightarrow SP$

## 5.21 Interrupt Instructions

Interrupt instructions handle transfer of control to a routine that performs a critical task. Software interrupts are a type of exception. **SECTION 7 EXCEPTION PROCESSING** covers interrupt exception processing in detail.

The SWI instruction initiates synchronous exception processing. First, the return PC value is stacked. After CPU context is stacked, execution continues at the address pointed to by the SWI vector.

Execution of the SWI instruction causes an interrupt without an interrupt service request. SWI is not inhibited by global mask bits I and X in the CCR, and execution of SWI sets the I mask bit. Once an SWI interrupt begins, maskable interrupts are inhibited until the I bit in the CCR is cleared. This typically occurs when an RTI instruction at the end of the SWI service routine restores context.

The CPU12 uses the software interrupt for unimplemented opcode trapping. There are opcodes in all 256 positions in the page 1 opcode map, but only 54 of the 256 positions on page 2 of the opcode map are used. If the CPU attempts to execute one of the unimplemented opcodes on page 2, an opcode trap interrupt occurs. Traps are essentially interrupts that share the \$FFF8:\$FFF9 interrupt vector.

The RTI instruction is used to terminate all exception handlers, including interrupt service routines. RTI first restores the CCR, B:A, X, Y, and the return address from the stack. If no other interrupt is pending, normal execution resumes with the instruction following the last instruction that executed prior to interrupt.

**Table 5-22** is a summary of interrupt instructions.

**Table 5-22 Interrupt Instructions**

Mnemonic	Function	Operation
RTI	Return from Interrupt	$(M_{(SP)}) \Rightarrow CCR; (SP) + \$0001 \Rightarrow SP$ $(M_{(SP)} : M_{(SP+1)}) \Rightarrow B : A; (SP) + \$0002 \Rightarrow SP$ $(M_{(SP)} : M_{(SP+1)}) \Rightarrow X_H : X_L; (SP) + \$0004 \Rightarrow SP$ $(M_{(SP)} : M_{(SP+1)}) \Rightarrow PC_H : PC_L; (SP) + \$0002 \Rightarrow SP$ $(M_{(SP)} : M_{(SP+1)}) \Rightarrow Y_H : Y_L; (SP) + \$0004 \Rightarrow SP$
SWI	Software Interrupt	$SP - 2 \Rightarrow SP; RTN_H : RTN_L \Rightarrow M_{(SP)} : M_{(SP+1)}$ $SP - 2 \Rightarrow SP; Y_H : Y_L \Rightarrow M_{(SP)} : M_{(SP+1)}$ $SP - 2 \Rightarrow SP; X_H : X_L \Rightarrow M_{(SP)} : M_{(SP+1)}$ $SP - 2 \Rightarrow SP; B : A \Rightarrow M_{(SP)} : M_{(SP+1)}$ $SP - 1 \Rightarrow SP; CCR \Rightarrow M_{(SP)}$
TRAP	Software Interrupt	$SP - 2 \Rightarrow SP; RTN_H : RTN_L \Rightarrow M_{(SP)} : M_{(SP+1)}$ $SP - 2 \Rightarrow SP; Y_H : Y_L \Rightarrow M_{(SP)} : M_{(SP+1)}$ $SP - 2 \Rightarrow SP; X_H : X_L \Rightarrow M_{(SP)} : M_{(SP+1)}$ $SP - 2 \Rightarrow SP; B : A \Rightarrow M_{(SP)} : M_{(SP+1)}$ $SP - 1 \Rightarrow SP; CCR \Rightarrow M_{(SP)}$

## 5.22 Index Manipulation Instructions

These instructions perform 8- and 16-bit operations on the three index registers and accumulators, other registers, or memory, as shown in **Table 5-23**.

**Table 5-23 Index Manipulation Instructions**

Addition Instructions		
Mnemonic	Function	Operation
ABX	Add B to X	$(B) + (X) \Rightarrow X$
ABY	Add B to Y	$(B) + (Y) \Rightarrow Y$
Compare Instructions		
Mnemonic	Function	Operation
CPS	Compare SP to Memory	$(SP) - (M : M + 1)$
CPX	Compare X to Memory	$(X) - (M : M + 1)$
CPY	Compare Y to Memory	$(Y) - (M : M + 1)$
Load Instructions		
Mnemonic	Function	Operation
LDS	Load SP from Memory	$M : M + 1 \Rightarrow SP$
LDX	Load X from Memory	$(M : M + 1) \Rightarrow X$
LDY	Load Y from Memory	$(M : M + 1) \Rightarrow Y$
LEAS	Load Effective Address into SP	Effective Address $\Rightarrow SP$
LEAX	Load Effective Address into X	Effective Address $\Rightarrow X$
LEAY	Load Effective Address into Y	Effective Address $\Rightarrow Y$
Store Instructions		
Mnemonic	Function	Operation
STS	Store SP in Memory	$(SP) \Rightarrow M : M + 1$
STX	Store X in Memory	$(X) \Rightarrow M : M + 1$
STY	Store Y in Memory	$(Y) \Rightarrow M : M + 1$
Transfer Instructions		
Mnemonic	Function	Operation
TFR	Transfer Register to Register	$(A, B, CCR, D, X, Y, \text{ or } SP) \Rightarrow A, B, CCR, D, X, Y, \text{ or } SP$
TSX	Transfer SP to X	$(SP) \Rightarrow X$
TSY	Transfer SP to Y	$(SP) \Rightarrow Y$
TXS	Transfer X to SP	$(X) \Rightarrow SP$
TYS	Transfer Y to SP	$(Y) \Rightarrow SP$
Exchange Instructions		
Mnemonic	Function	Operation
EXG	Exchange Register to Register	$(A, B, CCR, D, X, Y, \text{ or } SP) \Leftrightarrow (A, B, CCR, D, X, Y, \text{ or } SP)$
XGDX	EXchange D with X	$(D) \Leftrightarrow (X)$
XGDY	EXchange D with Y	$(D) \Leftrightarrow (Y)$

### 5.23 Stacking Instructions

There are two types of stacking instructions, as shown in **Table 5-24**. Stack pointer instructions use specialized forms of mathematical and data transfer instructions to perform stack pointer manipulation. Stack operation instructions save information on and retrieve information from the system stack.

**Table 5-24 Stacking Instructions**

Stack Pointer Instructions		
Mnemonic	Function	Operation
CPS	Compare SP to Memory	$(SP) - (M : M + 1)$
DES	Decrement SP	$(SP) - 1 \Rightarrow SP$
INS	Increment SP	$(SP) + 1 \Rightarrow SP$
LDS	Load SP	$(M : M + 1) \Rightarrow SP$
LEAS	Load Effective Address into SP	Effective Address $\Rightarrow SP$
STS	Store SP	$(SP) \Rightarrow M : M + 1$
TSX	Transfer SP to X	$(SP) \Rightarrow X$
TSY	Transfer SP to Y	$(SP) \Rightarrow Y$
TXS	Transfer X to SP	$(X) \Rightarrow SP$
TYS	Transfer Y to SP	$(Y) \Rightarrow SP$
Stack Operation Instructions		
Mnemonic	Function	Operation
PSHA	Push A	$(SP) - 1 \Rightarrow SP; (A) \Rightarrow M_{(SP)}$
PSHB	Push B	$(SP) - 1 \Rightarrow SP; (B) \Rightarrow M_{(SP)}$
PSHC	Push CCR	$(SP) - 1 \Rightarrow SP; (A) \Rightarrow M_{(SP)}$
PSHD	Push D	$(SP) - 2 \Rightarrow SP; (A : B) \Rightarrow M_{(SP)} : M_{(SP+1)}$
PSHX	Push X	$(SP) - 2 \Rightarrow SP; (X) \Rightarrow M_{(SP)} : M_{(SP+1)}$
PSHY	Push Y	$(SP) - 2 \Rightarrow SP; (Y) \Rightarrow M_{(SP)} : M_{(SP+1)}$
PULA	Pull A	$M_{(SP)} \Rightarrow A; (SP) + 1 \Rightarrow SP$
PULB	Pull B	$M_{(SP)} \Rightarrow B; (SP) + 1 \Rightarrow SP$
PULC	Pull CCR	$M_{(SP)} \Rightarrow CCR; (SP) + 1 \Rightarrow SP$
PULD	Pull D	$M_{(SP)} : M_{(SP+1)} \Rightarrow A : B; (SP) + 2 \Rightarrow SP$
PULX	Pull X	$M_{(SP)} : M_{(SP+1)} \Rightarrow X; (SP) + 2 \Rightarrow SP$
PULY	Pull Y	$M_{(SP)} : M_{(SP+1)} \Rightarrow Y; (SP) + 2 \Rightarrow SP$

### 5.24 Pointer and Index Calculation Instructions

The load effective address instructions allow 5-, 8-, or 16-bit constants, or the contents of 8-bit accumulators A and B or 16-bit accumulator D to be added to the contents of the X and Y index registers, the SP, or the PC. **Table 5-25** is a summary of pointer and index instructions.

**Table 5-25 Pointer and Index Calculation Instructions**

Mnemonic	Function	Operation
LEAS	Load Result of Indexed Addressing Mode Effective Address Calculation into Stack Pointer	$r \pm \text{Constant} \Rightarrow \text{SP}$ or $(r) + (\text{Accumulator}) \Rightarrow \text{SP}$ $r = \text{X, Y, SP, or PC}$
LEAX	Load Result of Indexed Addressing Mode Effective Address Calculation into X Index Register	$r \pm \text{Constant} \Rightarrow \text{X}$ or $(r) + (\text{Accumulator}) \Rightarrow \text{X}$ $r = \text{X, Y, SP, or PC}$
LEAY	Load Result of Indexed Addressing Mode Effective Address Calculation into Y Index Register	$r \pm \text{Constant} \Rightarrow \text{Y}$ or $(r) + (\text{Accumulator}) \Rightarrow \text{Y}$ $r = \text{X, Y, SP, or PC}$

### 5.25 Condition Code Instructions

Condition code instructions are special forms of mathematical and data transfer instructions that can be used to change the condition code register. **Table 5-26** shows instructions that can be used to manipulate the CCR.

**Table 5-26 Condition Codes Instructions**

Mnemonic	Function	Operation
ANDCC	Logical AND CCR with Memory	$(\text{CCR}) \bullet (\text{M}) \Rightarrow \text{CCR}$
CLC	Clear C Bit	$0 \Rightarrow \text{C}$
CLI	Clear I Bit	$0 \Rightarrow \text{I}$
CLV	Clear V Bit	$0 \Rightarrow \text{V}$
ORCC	Logical OR CCR with Memory	$(\text{CCR}) + (\text{M}) \Rightarrow \text{CCR}$
PSHC	Push CCR onto Stack	$(\text{SP}) - 1 \Rightarrow \text{SP}; (\text{CCR}) \Rightarrow \text{M}_{(\text{SP})}$
PULC	Pull CCR from Stack	$(\text{M}_{(\text{SP})}) \Rightarrow \text{CCR}; (\text{SP}) + 1 \Rightarrow \text{SP}$
SEC	Set C Bit	$1 \Rightarrow \text{C}$
SEI	Set I Bit	$1 \Rightarrow \text{I}$
SEV	Set V Bit	$1 \Rightarrow \text{V}$
TAP	Transfer A to CCR	$(\text{A}) \Rightarrow \text{CCR}$
TPA	Transfer CCR to A	$(\text{CCR}) \Rightarrow \text{A}$

### 5.26 Stop and Wait Instructions

As shown in **Table 5-27**, there are two instructions that put the CPU12 in an inactive state that reduces power consumption.

The stop instruction (STOP) stacks a return address and the contents of CPU registers and accumulators, then halts all system clocks.

The wait instruction (WAI) stacks a return address and the contents of CPU registers and accumulators, then waits for an interrupt service request; however, system clock signals continue to run.

Both STOP and WAI require that either an interrupt or a reset exception occur before normal execution of instructions resumes. Although both instructions require the same number of clock cycles to resume normal program execution after an interrupt service request is made, restarting after a STOP requires extra time for the oscillator to reach operating speed.

**Table 5-27 Stop and Wait Instructions**

Mnemonic	Function	Operation
STOP	Stop	$SP - 2 \Rightarrow SP; RTN_H : RTN_L \Rightarrow M_{(SP)} : M_{(SP+1)}$ $SP - 2 \Rightarrow SP; Y_H : Y_L \Rightarrow M_{(SP)} : M_{(SP+1)}$ $SP - 2 \Rightarrow SP; X_H : X_L \Rightarrow M_{(SP)} : M_{(SP+1)}$ $SP - 2 \Rightarrow SP; B : A \Rightarrow M_{(SP)} : M_{(SP+1)}$ $SP - 1 \Rightarrow SP; CCR \Rightarrow M_{(SP)}$ STOP CPU Clocks
WAI	Wait for Interrupt	$SP - 2 \Rightarrow SP; RTN_H : RTN_L \Rightarrow M_{(SP)} : M_{(SP+1)}$ $SP - 2 \Rightarrow SP; Y_H : Y_L \Rightarrow M_{(SP)} : M_{(SP+1)}$ $SP - 2 \Rightarrow SP; X_H : X_L \Rightarrow M_{(SP)} : M_{(SP+1)}$ $SP - 2 \Rightarrow SP; B : A \Rightarrow M_{(SP)} : M_{(SP+1)}$ $SP - 1 \Rightarrow SP; CCR \Rightarrow M_{(SP)}$

## 5.27 Background Mode and Null Operations

Background debug mode is a special CPU12 operating mode that is used for system development and debugging. Executing BGND when BDM is enabled puts the CPU12 in this mode. For complete information refer to **SECTION 8 DEVELOPMENT AND DEBUG SUPPORT**.

Null operations are often used to replace other instructions during software debugging. Replacing conditional branch instructions with BRN, for instance, permits testing a decision-making routine without actually taking the branches.

**Table 5-28** shows the BGND and NOP instructions.

**Table 5-28 Background Mode and Null Operation Instructions**

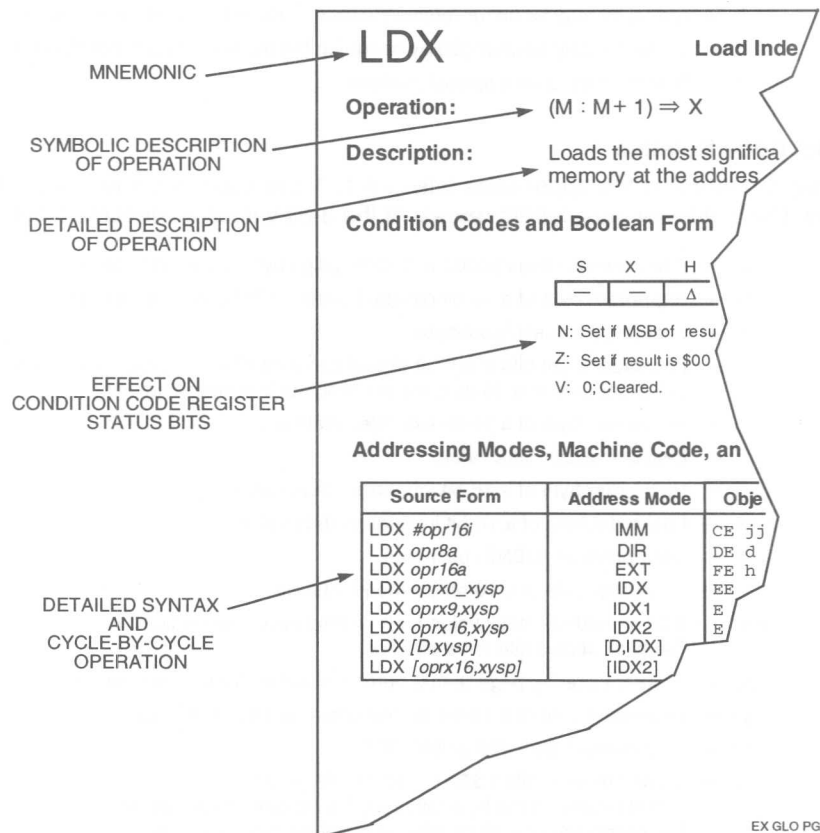
Mnemonic	Function	Operation
BGND	Enter Background Debug Mode	If BDM enabled, enter BDM; else, resume normal processing
BRN	Branch Never	Does not branch
LBRN	Long Branch Never	Does not branch
NOP	Null operation	—

## SECTION 6 INSTRUCTION GLOSSARY

This section is a comprehensive reference to the CPU12 instruction set.

### 6.1 Glossary Information

The glossary contains an entry for each assembler mnemonic, in alphabetic order. **Figure 6-1** is a representation of a glossary page.



**Figure 6-1 Example Glossary Page**



Each entry contains symbolic and textual descriptions of operation, information concerning the effect of operation on status bits in the condition code register, and a table that describes assembler syntax, cycle count, and cycle-by-cycle execution of the instruction.

## 6.2 Condition Code Changes

The following special characters are used to describe the effects of instruction execution on the status bits in the condition codes register.

- Status bit not affected by operation.
- 0 — Status bit cleared by operation.
- 1 — Status bit set by operation.
- Δ — Status bit affected by operation.
- ↓ — Status bit may be cleared or remain set, but is not set by operation.
- ↑ — Status bit may be set or remain cleared, but is not cleared by operation.
- ? — Status bit may be changed by operation but the final state is not defined.
- ! — Status bit used for a special purpose.

## 6.3 Object Code Notation

The digits 0 to 9 and the upper case letters A to F are used to express hexadecimal values. Pairs of lower case letters represent the 8-bit values as described below.

- dd — 8-bit direct address \$0000 to \$00FF. (High byte assumed to be \$00).
- ee — High-order byte of a 16-bit constant offset for indexed addressing.
- eb — Exchange/Transfer post-byte.
- ff — Low-order eight bits of a 9-bit signed constant offset for indexed addressing, or low-order byte of a 16-bit constant offset for indexed addressing.
- hh — High-order byte of a 16-bit extended address.
- ii — 8-bit immediate data value.
- jj — High-order byte of a 16-bit immediate data value.
- kk — Low-order byte of a 16-bit immediate data value.
- lb — Loop primitive (DBNE) post-byte.
- ll — Low-order byte of a 16-bit extended address.
- mm — 8-bit immediate mask value for bit manipulation instructions.  
Set bits indicate bits to be affected.
- pg — Program overlay page (bank) number used in CALL instruction.
- qq — High-order byte of a 16-bit relative offset for long branches.
- tn — Trap number \$30–\$39 or \$40–\$FF.
- rr — Signed relative offset \$80 (–128) to \$7F (+127).  
Offset relative to the byte following the relative offset byte, or low-order byte of a 16-bit relative offset for long branches.
- xb — Indexed addressing post-byte.

## 6.4 Source Forms

The glossary pages provide only essential information about assembler source forms. Assemblers generally support a number of assembler directives, allow definition of program labels, and have special conventions for comments. For complete information about writing source files for a particular assembler, refer to the documentation provided by the assembler vendor.

Assemblers are typically very flexible about the use of spaces and tabs. Often, any number of spaces or tabs can be used where a single space is shown on the glossary pages. Spaces and tabs are also normally allowed before and after commas. When program labels are used, there must also be at least one tab or space before all instruction mnemonics. This required space is not apparent in the source forms.

Everything in the source forms columns, *except expressions in italic characters*, is literal information which must appear in the assembly source file exactly as shown. The initial 3- to 5-letter mnemonic is always a literal expression. All commas, pound signs (#), parentheses, square brackets ([ or ]), plus signs (+), minus signs (–), and the register designation D (as in [D,...]), are literal characters.

Groups of italic characters in the columns represent variable information to be supplied by the programmer. These groups can include any alphanumeric character or the underscore character, but cannot include a space or comma. For example, the groups *xy* and *opr0\_xy* are both valid, but the two groups *opr0 xy* are not valid because there is a space between them. Permitted syntax is described below.

The definition of a legal label or expression varies from assembler to assembler. Assemblers also vary in the way CPU registers are specified. Refer to assembler documentation for detailed information. Recommended register designators are a, A, b, B, ccr, CCR, d, D, x, X, y, Y, sp, SP, pc, and PC.

- abc* — Any one legal register designator for accumulators A or B or the CCR.
- abcdxy* — Any one legal register designator for accumulators A or B, the CCR, the double accumulator D, index registers X or Y, or the SP. Some assemblers may accept t2, T2, t3, or T3 codes in certain cases of transfer and exchange instructions, but these forms are intended for Motorola use only.
- abd* — Any one legal register designator for accumulators A or B or the double accumulator D.
- abdxys* — Any one legal register designator for accumulators A or B, the double accumulator D, index register X or Y, or the SP.
- dxys* — Any one legal register designation for the double accumulator D, index registers X or Y, or the SP.
- msk8* — Any label or expression that evaluates to an 8-bit value. Some assemblers require a # symbol before this value.
- opr8i* — Any label or expression that evaluates to an 8-bit immediate value.
- opr16i* — Any label or expression that evaluates to a 16-bit immediate value.
- opr8a* — Any label or expression that evaluates to an 8-bit value. The instruction treats this 8-bit value as the low order 8-bits of an address in the direct page of the 64-Kbyte address space (\$00xx).

- opr16a* — Any label or expression that evaluates to a 16-bit value. The instruction treats this value as an address in the 64-Kbyte address space.
- opr0\_xysp* — This word breaks down into one of the following alternative forms that assemble to an 8-bit indexed addressing postbyte code. These forms generate the same object code except for the value of the postbyte code, which is designated as *xb* in the object code columns of the glossary pages. As with the source forms, treat all commas, plus signs, and minus signs as literal syntax elements. The italicized words used in these forms are included in this key.
- opr5,xysp*  
*opr3,-xys*  
*opr3,+xys*  
*opr3,xys-*  
*opr3,xys+*  
*abd,xysp*
- opr3* — Any label or expression that evaluates to a value in the range +1 to +8.
- opr5* — Any label or expression that evaluates to a 5-bit value in the range -16 to +15.
- opr9* — Any label or expression that evaluates to a 9-bit value in the range -256 to +255.
- opr16* — Any label or expression that evaluates to a 16-bit value. Since the CPU12 has a 16-bit address bus, this can be either a signed or an unsigned value.
- page* — Any label or expression that evaluates to an 8-bit value. The CPU12 recognizes up to an 8-bit page value for memory expansion but not all MCUs that include the CPU12 implement all of these bits. It is the programmer's responsibility to limit the page value to legal values for the intended MCU system. Some assemblers require a # symbol before this value.
- rel8* — Any label or expression that refers to an address that is within -256 to +255 locations from the next address after the last byte of object code for the current instruction. The assembler will calculate the 8-bit signed offset and include it in the object code for this instruction.
- rel9* — Any label or expression that refers to an address that is within -512 to +511 locations from the next address after the last byte of object code for the current instruction. The assembler will calculate the 9-bit signed offset and include it in the object code for this instruction. The sign bit for this 9-bit value is encoded by the assembler as a bit in the looping postbyte (*lb*) of one of the loop control instructions DBEQ, DBNE, IBNE, TBEQ, or TBNE. The remaining eight bits of the offset are included as an extra byte of object code.
- rel16* — Any label or expression that refers to an address anywhere in the 64-Kbyte address space. The assembler will calculate the 16-bit signed offset between this address and the next address after the last byte of object code for this instruction, and include it in the object code for this instruction.
- trapnum* — Any label or expression that evaluates to an 8-bit number in the range \$30-\$39 or \$40-\$FF. Used for TRAP instruction.
- xys* — Any one legal register designation for index registers X or Y or the SP.
- xysp* — Any one legal register designation for index registers X or Y, the SP, or the PC. The reference point for PC relative instructions is the next address after the last byte of object code for the current instruction.

## 6.5 Cycle-by-Cycle Execution

This information is found in the tables at the bottom of each instruction glossary page. Entries show how many bytes of information are accessed from different areas of memory during the course of instruction execution. With this information and knowledge of the type and speed of memory in the system, a user can determine the execution time for any instruction in any system.

A single letter code in the column represents a single CPU cycle. Upper case letters indicate 16-bit access cycles. There are cycle codes for each addressing mode variation of each instruction. Simply count code letters to determine the execution time of an instruction in a best-case system. An example of a best-case system is a single-chip 16-bit system with no 16-bit off-boundary data accesses to any locations other than on-chip RAM.

Many conditions can cause one or more instruction cycles to be stretched, but the CPU is not aware of the stretch delays because the clock to the CPU is temporarily stopped during these delays.

The following paragraphs explain the cycle code letters used and note conditions that can cause each type of cycle to be stretched.

- f — Free cycle. This indicates a cycle where the CPU does not require use of the system buses. An f cycle is always one cycle of the system bus clock. These cycles can be used by a queue controller or the background debug system to perform single cycle accesses without disturbing the CPU.
- g — Read 8-bit PPAGE register. These cycles are only used with the CALL instruction to read the current value of the PPAGE register, and are not visible on the external bus. Since the PPAGE register is an internal 8-bit register, these cycles are never stretched.
- l — Read indirect pointer. Indexed indirect instructions use this 16-bit pointer from memory to address the operand for the instruction. These are always 16-bit reads but they can be either aligned or misaligned. These cycles are extended to two bus cycles if the MCU is operating with an 8-bit external data bus and the corresponding data is stored in external memory. There can be additional stretching when the address space is assigned to a chip-select circuit programmed for slow memory. These cycles are also stretched if they correspond to misaligned access to a memory that is not designed for single-cycle misaligned access.
- i — Read indirect PPAGE value. These cycles are only used with indexed indirect versions of the CALL instruction, where the 8-bit value for the memory expansion page register of the CALL destination is fetched from an indirect memory location. These cycles are stretched only when controlled by a chip-select circuit that is programmed for slow memory.
- n — Write 8-bit PPAGE register. These cycles are only used with the CALL and RTC instructions to write the destination value of the PPAGE register and are not visible on the external bus. Since the PPAGE register is an internal 8-bit register, these cycles are never stretched.

- O — Optional cycle. Program information is always fetched as aligned 16-bit words. When an instruction consists of an odd number of bytes, and the first byte is misaligned, an O cycle is used to make an additional program word access (P) cycle that maintains queue order. In all other cases, the O cycle appears as a free (f) cycle. The \$18 prebyte for page two opcodes is treated as a special one-byte instruction. If the prebyte is misaligned, the O cycle is used as a program word access for the prebyte; if the prebyte is aligned, the O cycle appears as a free cycle. If the remainder of the instruction consists of an odd number of bytes, another O cycle is required some time before the instruction is completed. If the O cycle for the prebyte is treated as a P cycle, any subsequent O cycle in the same instruction is treated as an f cycle; if the O cycle for the prebyte is treated as an f cycle, any subsequent O cycle in the same instruction is treated as a P cycle. Optional cycles used for program word accesses can be extended to two bus cycles if the MCU is operating with an 8-bit external data bus and the program is stored in external memory. There can be additional stretching when the address space is assigned to a chip-select circuit programmed for slow memory. Optional cycles used as free cycles are never stretched.
- P — Program word access. Program information is fetched as aligned 16-bit words. These cycles are extended to two bus cycles if the MCU is operating with an 8-bit external data bus and the program is stored externally. There can be additional stretching when the address space is assigned to a chip-select circuit programmed for slow memory.
- r — 8-bit data read. These cycles are stretched only when controlled by a chip-select circuit programmed for slow memory.
- R — 16-bit data read. These cycles are extended to two bus cycles if the MCU is operating with an 8-bit external data bus and the corresponding data is stored in external memory. There can be additional stretching when the address space is assigned to a chip-select circuit programmed for slow memory. These cycles are also stretched if they correspond to misaligned accesses to memory that is not designed for single-cycle misaligned access.
- s — Stack 8-bit data. These cycles are stretched only when controlled by a chip-select circuit programmed for slow memory.
- S — Stack 16-bit data. These cycles are extended to two bus cycles if the MCU is operating with an 8-bit external data bus and the SP is pointing to external memory. There can be additional stretching if the address space is assigned to a chip-select circuit programmed for slow memory. These cycles are also stretched if they correspond to misaligned accesses to a memory that is not designed for single cycle misaligned access. The internal RAM is designed to allow single cycle misaligned word access.
- w — 8-bit data write. These cycles are stretched only when controlled by a chip-select circuit programmed for slow memory.
- W — 16-bit data write. These cycles are extended to two bus cycles if the MCU is operating with an 8-bit external data bus and the corresponding data is stored in external memory. There can be additional stretching when the address space is assigned to a chip-select circuit programmed for slow memory. These cycles are also stretched if they correspond to misaligned access to a memory that is not designed for single-cycle misaligned access.
- u — Unstack 8-bit data. These cycles are stretched only when controlled by a chip-select circuit programmed for slow memory.

- U — Unstack 16-bit data. These cycles are extended to two bus cycles if the MCU is operating with an 8-bit external data bus and the SP is pointing to external memory. There can be additional stretching when the address space is assigned to a chip-select circuit programmed for slow memory. These cycles are also stretched if they correspond to misaligned accesses to a memory that is not designed for single-cycle misaligned access. The internal RAM is designed to allow single-cycle misaligned word access.
- V — Vector fetch. Vectors are always aligned 16-bit words. These cycles are extended to two bus cycles if the MCU is operating with an 8-bit external data bus and the program is stored in external memory. There can be additional stretching when the address space is assigned to a chip-select circuit programmed for slow memory.
- t — 8-bit conditional read. These cycles are either data read cycles or free cycles, depending upon the data and flow of the REVW instruction. These cycles are only stretched when controlled by a chip-select circuit programmed for slow memory.
- T — 16-bit conditional read. These cycles are either data read cycles or free cycles, depending upon the data and flow of the REV or REVW instruction. These cycles are extended to two bus cycles if the MCU is operating with an 8-bit external data bus and the corresponding data is stored in external memory. There can be additional stretching when the address space is assigned to a chip-select circuit programmed for slow memory. These cycles are also stretched if they correspond to misaligned accesses to a memory that is not designed for single-cycle misaligned access.
- x — 8-bit conditional write. These cycles are either data write cycles or free cycles, depending upon the data and flow of the REV or REVW instruction. These cycles are only stretched when controlled by a chip-select circuit programmed for slow memory.

#### Special Notation for Branch Taken/Not Taken Cases

- PPP/P — Short branches require three cycles if taken, one cycle if not taken. Since the instruction consists of a single word containing both an opcode and an 8-bit offset, the not-taken case is simple — the queue advances, another program word fetch is made, and execution continues with the next instruction. The taken case requires that the queue be refilled so that execution can continue at a new address. First, the effective address of the destination is determined, then the CPU performs three program word fetches from that address.
- OPPP/OPO — Long branches require four cycles if taken, three cycles if not taken. Optional cycles are required because all long branches are page two opcodes, and thus include the \$18 prebyte. The CPU12 treats the prebyte as a special 1-byte instruction. If the prebyte is misaligned, the optional cycle is used to perform a program word access; if the prebyte is aligned, the optional cycle is used to perform a free cycle. As a result, both the taken and not-taken cases use one optional cycle for the prebyte. In the not-taken case, the queue must advance so that execution can continue with the next instruction, and another optional cycle is required to maintain the queue. The taken case requires that the queue be refilled so that execution can continue at a new address. First, the effective address of the destination is determined, then the CPU performs three program word fetches from that address.

## 6.6 Glossary

# ABA

**Add Accumulator B To  
Accumulator A**

# ABA

**Operation:**  $(A) + (B) \Rightarrow A$

**Description:** Adds the content of accumulator B to the content of accumulator A and places the result in A. The content of B is not changed. This instruction affects the H status bit so it is suitable for use in BCD arithmetic operations (see DAA instruction for additional information).

**Condition Codes and Boolean Formulas:**

S	X	H	I	N	Z	V	C
-	-	$\Delta$	-	$\Delta$	$\Delta$	$\Delta$	$\Delta$

H:  $A3 \bullet B3 + B3 \bullet \overline{R3} + \overline{R3} \bullet A3$

Set if there was a carry from bit 3; cleared otherwise.

N: Set if MSB of result is set; cleared otherwise.

Z: Set if result is \$00; cleared otherwise.

V:  $A7 \bullet B7 \bullet \overline{R7} + \overline{A7} \bullet \overline{B7} \bullet R7$

Set if a two's complement overflow resulted from the operation; cleared otherwise.

C:  $A7 \bullet B7 + B7 \bullet \overline{R7} + \overline{R7} \bullet A7$

Set if there was a carry from the MSB of the result; cleared otherwise.

**Addressing Modes, Machine Code, and Execution Times:**

Source Form	Address Mode	Object Code	Cycles	Access Detail
ABA	INH	18 06	2	00

# ABX

Add Accumulator B to  
Index Register X

# ABX

**Operation:** (B) + (X)  $\Rightarrow$  X

**Description:** Adds the 8-bit unsigned content of accumulator B to the content of index register X considering the possible carry out of the low-order byte of X; places the result in X. The content of B is not changed.

This mnemonic is implemented by the LEAX B,X instruction. The LEAX instruction allows A, B, D, or a constant to be added to X. For compatibility with the M68HC11, the mnemonic ABX is translated into the LEAX B,X instruction by the assembler.

**Condition Codes and Boolean Formulas:**

S	X	H	I	N	Z	V	C
-	-	-	-	-	-	-	-

None affected.

**Addressing Modes, Machine Code, and Execution Times:**

Source Form	Address Mode	Object Code	Cycles	Access Detail
ABX translates to... LEAX B,X	IDX	1A E5	2	pp1

Notes:

1. Due to internal CPU requirements, the program word fetch is performed twice to the same address during this instruction.



# ABY

## Add Accumulator B to Index Register Y

# ABY

**Operation:** (B) + (Y)  $\Rightarrow$  Y

**Description:** Adds the 8-bit unsigned content of accumulator B to the content of index register Y considering the possible carry out of the low-order byte of Y; places the result in Y. The content of B is not changed.

This mnemonic is implemented by the LEAY B,Y instruction. The LEAY instruction allows A, B, D, or a constant to be added to Y. For compatibility with the M68HC11, the mnemonic ABY is translated into the LEAY B,Y instruction by the assembler.

### Condition Codes and Boolean Formulas:

S	X	H	I	N	Z	V	C
-	-	-	-	-	-	-	-

None affected.

### Addressing Modes, Machine Code, and Execution Times:

Source Form	Address Mode	Object Code	Cycles	Access Detail
ABY translates to... LEAY B,Y	IDX	19 ED	2	pp1

Notes:

1. Due to internal CPU requirements, the program word fetch is performed twice to the same address during this instruction.

# ADCA

Add with Carry to A

# ADCA

**Operation:**  $(A) + (M) + C \Rightarrow A$

**Description:** Adds the content of accumulator A to the content of memory location M, then adds the value of the C bit and places the result in A. This instruction affects the H status bit, so it is suitable for use in BCD arithmetic operations (see DAA instruction for additional information).

## Condition Codes and Boolean Formulas:

S	X	H	I	N	Z	V	C
—	—	$\Delta$	—	$\Delta$	$\Delta$	$\Delta$	$\Delta$

- H:  $X3 \bullet M3 + M3 \bullet \overline{R3} + \overline{R3} \bullet X3$   
Set if there was a carry from bit 3; cleared otherwise.
- N: Set if MSB of result is set; cleared otherwise.
- Z: Set if result is \$00; cleared otherwise.
- V:  $X7 \bullet M7 \bullet \overline{R7} + \overline{X7} \bullet \overline{M7} \bullet R7$   
Set if two's complement overflow resulted from the operation; cleared otherwise.
- C:  $X7 \bullet M7 + M7 \bullet \overline{R7} + \overline{R7} \bullet X7$   
Set if there was a carry from the MSB of the result; cleared otherwise.

## Addressing Modes, Machine Code, and Execution Times:

Source Form	Address Mode	Object Code	Cycles	Access Detail
ADCA #opr8i	IMM	89 ii	1	P
ADCA opr8a	DIR	99 dd	3	rFP
ADCA opr16a	EXT	B9 hh ll	3	rOP
ADCA oprx0_xysp	IDX	A9 xb	3	rFP
ADCA oprx9_xysp	IDX1	A9 xb ff	3	rPO
ADCA oprx16_xysp	IDX2	A9 xb ee ff	4	frPP
ADCA [D,xysp]	[D,IDX]	A9 xb	6	fIfrFP
ADCA [oprx16,xysp]	[IDX2]	A9 xb ee ff	6	fIPrFP

# ADCB

Add with Carry to B

# ADCB

**Operation:**  $(B) + (M) + C \Rightarrow B$

**Description:** Adds the content of accumulator B to the content of memory location M, then adds the value of the C bit and places the result in B. This instruction affects the H status bit, so it is suitable for use in BCD arithmetic operations (see DAA instruction for additional information).

**Condition Codes and Boolean Formulas:**

S	X	H	I	N	Z	V	C
-	-	$\Delta$	-	$\Delta$	$\Delta$	$\Delta$	$\Delta$

- H:  $X3 \bullet M3 + M3 \bullet \overline{R3} + \overline{R3} \bullet X3$   
Set if there was a carry from bit 3; cleared otherwise.
- N: Set if MSB of result is set; cleared otherwise.
- Z: Set if result is \$00; cleared otherwise.
- V:  $X7 \bullet M7 \bullet \overline{R7} + \overline{X7} \bullet \overline{M7} \bullet R7$   
Set if two's complement overflow resulted from the operation; cleared otherwise.
- C:  $X7 \bullet M7 + M7 \bullet \overline{R7} + \overline{R7} \bullet X7$   
Set if there was a carry from the MSB of the result; cleared otherwise.

**Addressing Modes, Machine Code, and Execution Times:**

Source Form	Address Mode	Object Code	Cycles	Access Detail
ADCB #opr8i	IMM	C9 ii	1	P
ADCB opr8a	DIR	D9 dd	3	rFP
ADCB opr16a	EXT	F9 hh ll	3	rOP
ADCB oprx0_xysp	IDX	E9 xb	3	rFP
ADCB oprx9,xysp	IDX1	E9 xb ff	3	rPO
ADCB oprx16,xysp	IDX2	E9 xb ee ff	4	frPP
ADCB [D,xysp]	[D,IDX]	E9 xb	6	fIfFrFP
ADCB [oprx16,xysp]	[IDX2]	E9 xb ee ff	6	fIPrFP

# ADDA

Add without Carry to A

# ADDA

**Operation:**  $(A) + (M) \Rightarrow A$

**Description:** Adds the content of memory location M to accumulator A and places the result in A. This instruction affects the H status bit, so it is suitable for use in BCD arithmetic operations (see DAA instruction for additional information).

**Condition Codes and Boolean Formulas:**

S	X	H	I	N	Z	V	C
—	—	$\Delta$	—	$\Delta$	$\Delta$	$\Delta$	$\Delta$

- H:  $X3 \bullet M3 + M3 \bullet \overline{R3} + \overline{R3} \bullet X3$   
Set if there was a carry from bit 3; cleared otherwise.
- N: Set if MSB of result is set; cleared otherwise.
- Z: Set if result is \$00; cleared otherwise.
- V:  $X7 \bullet M7 \bullet \overline{R7} + \overline{X7} \bullet \overline{M7} \bullet R7$   
Set if two's complement overflow resulted from the operation; cleared otherwise.
- C:  $X7 \bullet M7 + M7 \bullet \overline{R7} + \overline{R7} \bullet X7$   
Set if there was a carry from the MSB of the result; cleared otherwise.

**Addressing Modes, Machine Code, and Execution Times:**

Source Form	Address Mode	Object Code	Cycles	Access Detail
ADDA #opr8i	IMM	8B ii	1	P
ADDA opr8a	DIR	9B dd	3	rFP
ADDA opr16a	EXT	BB hh ll	3	rOP
ADDA oprx0,xysp	IDX	AB xb	3	rFP
ADDA oprx9,xysp	IDX1	AB xb ff	3	rPO
ADDA oprx16,xysp	IDX2	AB xb ee ff	4	frPP
ADDA [D,xysp]	[D,IDX]	AB xb	6	fIfFrFP
ADDA [oprx16,xysp]	[IDX2]	AB xb ee ff	6	fIPrFP

# ADDB

Add without Carry to B

# ADDB

**Operation:** (B) + (M) ⇒ B

**Description:** Adds the content of memory location M to accumulator B and places the result in B. This instruction affects the H status bit, so it is suitable for use in BCD arithmetic operations (see DAA instruction for additional information).

**Condition Codes and Boolean Formulas:**

S	X	H	I	N	Z	V	C
-	-	Δ	-	Δ	Δ	Δ	Δ

- H:  $X3 \bullet M3 + M3 \bullet \overline{R3} + \overline{R3} \bullet X3$   
Set if there was a carry from bit 3; cleared otherwise.
- N: Set if MSB of result is set; cleared otherwise.
- Z: Set if result is \$00; cleared otherwise.
- V:  $X7 \bullet M7 \bullet \overline{R7} + \overline{X7} \bullet \overline{M7} \bullet R7$   
Set if two's complement overflow resulted from the operation; cleared otherwise.
- C:  $X7 \bullet M7 + M7 \bullet \overline{R7} + \overline{R7} \bullet X7$   
Set if there was a carry from the MSB of the result; cleared otherwise.

**Addressing Modes, Machine Code, and Execution Times:**

Source Form	Address Mode	Object Code	Cycles	Access Detail
ADDB #opr8i	IMM	CB ii	1	P
ADDB opr8a	DIR	DB dd	3	rFP
ADDB opr16a	EXT	FB hh ll	3	rOP
ADDB oprx0_xysp	IDX	EB xb	3	rFP
ADDB oprx9_xysp	IDX1	EB xb ff	3	rPO
ADDB oprx16_xysp	IDX2	EB xb ee ff	4	frPP
ADDB [D,xysp]	[D,IDX]	EB xb	6	fIfFP
ADDB [oprx16,xysp]	[IDX2]	EB xb ee ff	6	fIPrFP

# ADDD

## Add Double Accumulator

# ADDD

**Operation:**  $(A : B) + (M : M+1) \Rightarrow A : B$

**Description:** Adds the content of memory location M concatenated with the content of memory location M +1 to the content of double accumulator D and places the result in D. Accumulator A forms the high-order half of 16-bit double accumulator D; accumulator B forms the low-order half.

**Condition Codes and Boolean Formulas:**

S	X	H	I	N	Z	V	C
—	—	—	—	$\Delta$	$\Delta$	$\Delta$	$\Delta$

N: Set if MSB of result is set; cleared otherwise.

Z: Set if result is \$0000; cleared otherwise.

V:  $D15 \bullet M15 \bullet \overline{R15} + \overline{D15} \bullet \overline{M15} \bullet R15$   
Set if two's complement overflow resulted from the operation; cleared otherwise.

C:  $D15 \bullet M15 + M15 \bullet \overline{R15} + \overline{R15} \bullet D15$   
Set if there was a carry from the MSB of the result; cleared otherwise.

**Addressing Modes, Machine Code, and Execution Times:**

Source Form	Address Mode	Object Code	Cycles	Access Detail
ADDD #opr16i	IMM	C3 jj kk	2	OP
ADDD opr8a	DIR	D3 dd	3	RfP
ADDD opr16a	EXT	F3 hh ll	3	ROP
ADDD oprx0_xysp	IDX	E3 xb	3	RfP
ADDD oprx9,xysp	IDX1	E3 xb ff	3	RPO
ADDD oprx16,xysp	IDX2	E3 xb ee ff	4	fRPP
ADDD [D,xysp]	[D,IDX]	E3 xb	6	fIfRfP
ADDD [oprx16,xysp]	[IDX2]	E3 xb ee ff	6	fIPRfP

# ANDA

Logical AND A

# ANDA

Operation:  $(A) \bullet (M) \Rightarrow A$

**Description:** Performs logical AND between the content of memory location M and the content of accumulator A. The result is placed in A. After the operation is performed, each bit of A is the logical AND of the corresponding bits of M and of A before the operation began.

**Condition Codes and Boolean Formulas:**

S	X	H	I	N	Z	V	C
-	-	-	-	$\Delta$	$\Delta$	0	-

N: Set if MSB of result is set; cleared otherwise.

Z: Set if result is \$00; cleared otherwise.

V: 0; Cleared.

**Addressing Modes, Machine Code, and Execution Times:**

Source Form	Address Mode	Object Code	Cycles	Access Detail
ANDA #opr8i	IMM	84 ii	1	P
ANDA opr8a	DIR	94 dd	3	rfrP
ANDA opr16a	EXT	B4 hh ll	3	rOP
ANDA oprx0_xysp	IDX	A4 xb	3	rfrP
ANDA oprx9_xysp	IDX1	A4 xb ff	3	rPO
ANDA oprx16_xysp	IDX2	A4 xb ee ff	4	frPP
ANDA [D,xysp]	[D,IDX]	A4 xb	6	fIfrrfP
ANDA [opr16,xysp]	[IDX2]	A4 xb ee ff	6	fIPrfP

# ANDB

Logical AND B

# ANDB

**Operation:** (B) • (M) ⇒ B

**Description:** Performs logical AND between the content of memory location M and the content of accumulator B. The result is placed in B. After the operation is performed, each bit of B is the logical AND of the corresponding bits of M and of B before the operation began.

**Condition Codes and Boolean Formulas:**

S	X	H	I	N	Z	V	C
-	-	-	-	Δ	Δ	0	-

N: Set if MSB of result is set; cleared otherwise.

Z: Set if result is \$00; cleared otherwise.

V: 0; Cleared.

**Addressing Modes, Machine Code, and Execution Times:**

Source Form	Address Mode	Object Code	Cycles	Access Detail
ANDB #opr8i	IMM	C4 ii	1	P
ANDB opr8a	DIR	D4 dd	3	rfP
ANDB opr16a	EXT	F4 hh ll	3	rOP
ANDB oprx0_xysp	IDX	E4 xb	3	rfP
ANDB oprx9_xysp	IDX1	E4 xb ff	3	rPO
ANDB oprx16_xysp	IDX2	E4 xb ee ff	4	frPP
ANDB [D,xysp]	[D,IDX]	E4 xb	6	fIfrfP
ANDB [oprx16,xysp]	[IDX2]	E4 xb ee ff	6	fIPrfP



ANDCC

Logical AND CCR with Mask

ANDCC

**Operation:** (CCR) • (Mask) ⇒ CCR

**Description:** Performs bitwise logical AND between the content of a mask operand and the content of the CCR. The result is placed in the CCR. After the operation is performed, each bit of the CCR is the result of a logical AND with the corresponding bits of the mask. To clear CCR bits, clear the corresponding mask bits. CCR bits that correspond to ones in the mask are not changed by the ANDCC operation.

If the I mask bit is cleared, there is a one cycle delay before the system allows interrupt requests. This prevents interrupts from occurring between instructions in the sequences CLI, WAI and CLI, SEI (CLI is equivalent to ANDCC #\$EF).

**Condition Codes and Boolean Formulas:**

S	X	H	I	N	Z	V	C
↓	↓	↓	↓	↓	↓	↓	↓

Condition code bits are cleared if the corresponding bit was zero before the operation or if the corresponding bit in the mask is zero.

**Addressing Modes, Machine Code, and Execution Times:**

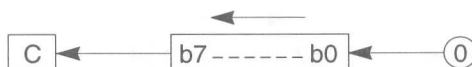
Source Form	Address Mode	Object Code	Cycles	Access Detail
ANDCC #opr8i	IMM	10 ii	1	P

# ASL

## Arithmetic Shift Left Memory (same as LSL)

# ASL

### Operation:



**Description:** Shifts all bits of memory location M one bit position to the left. Bit 0 is loaded with a zero. The C status bit is loaded from the most significant bit of M.

### Condition Codes and Boolean Formulas:

S	X	H	I	N	Z	V	C
—	—	—	—	Δ	Δ	Δ	Δ

N: Set if MSB of result is set; cleared otherwise.

Z: Set if result is \$00; cleared otherwise.

V:  $N \oplus C = [N \bullet \overline{C}] + [\overline{N} \bullet C]$  (for N and C after the shift)

Set if (N is set and C is cleared) or (N is cleared and C is set); cleared otherwise (for values of N and C after the shift).

C: M7

Set if the MSB of M was set before the shift; cleared otherwise.

### Addressing Modes, Machine Code, and Execution Times:

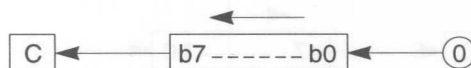
Source Form	Address Mode	Object Code	Cycles	Access Detail
ASL <i>opr16a</i>	EXT	78 hh ll	4	rOPw
ASL <i>opr0_xysp</i>	IDX	68 xb	3	rPw
ASL <i>opr9_xysp</i>	IDX1	68 xb ff	4	rPOw
ASL <i>opr16_xysp</i>	IDX2	68 xb ee ff	5	frPPw
ASL [D, <i>xysp</i> ]	[D,IDX]	68 xb	6	fIfPrPw
ASL [ <i>opr16_xysp</i> ]	[IDX2]	68 xb ee ff	6	fIPrPw

# ASLA

Arithmetic Shift Left A  
(same as LSLA)

# ASLA

Operation:



**Description:** Shifts all bits of accumulator A one bit position to the left. Bit 0 is loaded with a zero. The C status bit is loaded from the most significant bit of A.

**Condition Codes and Boolean Formulas:**

S	X	H	I	N	Z	V	C
-	-	-	-	Δ	Δ	Δ	Δ

N: Set if MSB of result is set; cleared otherwise.

Z: Set if result is \$00; cleared otherwise.

V:  $N \oplus C = [N \cdot \bar{C}] + [\bar{N} \cdot C]$  (for N and C after the shift)  
Set if (N is set and C is cleared) or (N is cleared and C is set); cleared otherwise (for values of N and C after the shift).

C: A7  
Set if the MSB of A was set before the shift; cleared otherwise.

**Addressing Modes, Machine Code, and Execution Times:**

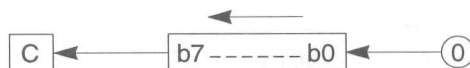
Source Form	Address Mode	Object Code	Cycles	Access Detail
ASLA	INH	48	1	0

# ASLB

Arithmetic Shift Left B  
(same as LSLB)

# ASLB

## Operation:



**Description:** Shifts all bits of accumulator B one bit position to the left. Bit 0 is loaded with a zero. The C status bit is loaded from the most significant bit of B.

## Condition Codes and Boolean Formulas:

S	X	H	I	N	Z	V	C
-	-	-	-	$\Delta$	$\Delta$	$\Delta$	$\Delta$

N: Set if MSB of result is set; cleared otherwise.

Z: Set if result is \$00; cleared otherwise.

V:  $N \oplus C = [N \bullet \bar{C}] + [\bar{N} \bullet C]$  (for N and C after the shift)  
Set if (N is set and C is cleared) or (N is cleared and C is set); cleared otherwise (for values of N and C after the shift).

C: B7  
Set if the MSB of B was set before the shift; cleared otherwise.

## Addressing Modes, Machine Code, and Execution Times:

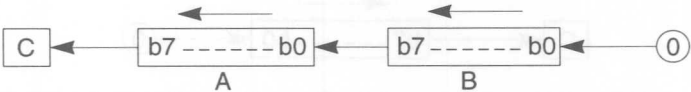
Source Form	Address Mode	Object Code	Cycles	Access Detail
ASLB	INH	58	1	0

ASLD

Arithmetic Shift Left Double Accumulator  
(same as LSLD)

ASLD

Operation:



**Description:** Shifts all bits of double accumulator D one bit position to the left. Bit 0 is loaded with a zero. The C status bit is loaded from the most significant bit of D.

Condition Codes and Boolean Formulas:

S	X	H	I	N	Z	V	C
-	-	-	-	Δ	Δ	Δ	Δ

- N: Set if MSB of result is set; cleared otherwise.
- Z: Set if result is \$0000; cleared otherwise.
- V:  $N \oplus C = [N \bullet \bar{C}] + [\bar{N} \bullet C]$  (for N and C after the shift)  
Set if (N is set and C is cleared) or (N is cleared and C is set); cleared otherwise (for values of N and C after the shift).
- C: D15  
Set if the MSB of D was set before the shift; cleared otherwise.

Addressing Modes, Machine Code, and Execution Times:

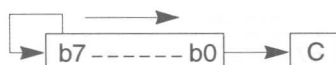
Source Form	Address Mode	Object Code	Cycles	Access Detail
ASLD	INH	59	1	o

# ASR

## Arithmetic Shift Right Memory

# ASR

### Operation:



**Description:** Shifts all bits of memory location M one place to the right. Bit 7 is held constant. Bit 0 is loaded into the C status bit. This operation effectively divides a two's complement value by two without changing its sign. The carry bit can be used to round the result.

### Condition Codes and Boolean Formulas:

S	X	H	I	N	Z	V	C
-	-	-	-	$\Delta$	$\Delta$	$\Delta$	$\Delta$

N: Set if MSB of result is set; cleared otherwise.

Z: Set if result is \$00; cleared otherwise.

V:  $N \oplus C = [N \bullet \bar{C}] + [\bar{N} \bullet C]$  (for N and C after the shift)

Set if (N is set and C is cleared) or (N is cleared and C is set); cleared otherwise (for values of N and C after the shift).

C: M0

Set if the LSB of M was set before the shift; cleared otherwise.

### Addressing Modes, Machine Code, and Execution Times:

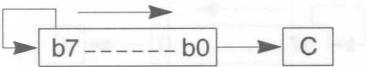
Source Form	Address Mode	Object Code	Cycles	Access Detail
ASR <i>opr16a</i>	EXT	77 hh ll	4	rOPw
ASR <i>opr0_xysp</i>	IDX	67 xb	3	rPw
ASR <i>opr9,xysp</i>	IDX1	67 xb ff	4	rPOw
ASR <i>opr16,xysp</i>	IDX2	67 xb ee ff	5	frPPw
ASR [D, <i>xysp</i> ]	[D,IDX]	67 xb	6	fiFrPw
ASR [ <i>opr16,xysp</i> ]	[IDX2]	67 xb ee ff	6	fiPrPw

# ASRA

Arithmetic Shift Right A

# ASRA

**Operation:**



**Description:** Shifts all bits of accumulator A one place to the right. Bit 7 is held constant. Bit 0 is loaded into the C status bit. This operation effectively divides a two's complement value by two without changing its sign. The carry bit can be used to round the result.

**Condition Codes and Boolean Formulas:**

S	X	H	I	N	Z	V	C
-	-	-	-	$\Delta$	$\Delta$	$\Delta$	$\Delta$

- N: Set if MSB of result is set; cleared otherwise.
- Z: Set if result is \$00; cleared otherwise.
- V:  $N \oplus C = [N \bullet \bar{C}] + [\bar{N} \bullet C]$  (for N and C after the shift)  
Set if (N is set and C is cleared) or (N is cleared and C is set); cleared otherwise (for values of N and C after the shift).
- C: A0  
Set if the LSB of A was set before the shift; cleared otherwise.

**Addressing Modes, Machine Code, and Execution Times:**

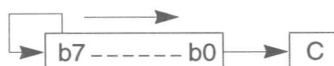
Source Form	Address Mode	Object Code	Cycles	Access Detail
ASRA	INH	47	1	0

# ASRB

## Arithmetic Shift Right B

# ASRB

### Operation:



**Description:** Shifts all bits of accumulator B one place to the right. Bit 7 is held constant. Bit 0 is loaded into the C status bit. This operation effectively divides a two's complement value by two without changing its sign. The carry bit can be used to round the result.

### Condition Codes and Boolean Formulas:

S	X	H	I	N	Z	V	C
—	—	—	—	Δ	Δ	Δ	Δ

N: Set if MSB of result is set; cleared otherwise.

Z: Set if result is \$00; cleared otherwise.

V:  $N \oplus C = [N \bullet \bar{C}] + [\bar{N} \bullet C]$  (for N and C after the shift)  
Set if (N is set and C is cleared) or (N is cleared and C is set); cleared otherwise (for values of N and C after the shift).

C: B0  
Set if the LSB of B was set before the shift; cleared otherwise.

### Addressing Modes, Machine Code, and Execution Times:

Source Form	Address Mode	Object Code	Cycles	Access Detail
ASRB	INH	57	1	0



# BCC

Branch if Carry Cleared  
(Same as BHS)

# BCC

**Operation:** If  $C = 0$ , then  $(PC) + \$0002 + Rel \Rightarrow PC$

Simple branch

**Description:** Tests the C status bit and branches if  $C = 0$ .

See **3.7 Relative Addressing Mode** for details of branch execution.

**Condition Codes and Boolean Formulas:**

S	X	H	I	N	Z	V	C
—	—	—	—	—	—	—	—

None affected.

**Addressing Modes, Machine Code, and Execution Times:**

Source Form	Address Mode	Object Code	Cycles	Access Detail
BCC <i>rel8</i>	REL	24 rr	3/1	PPP/P <sup>1</sup>

Notes:

1. PPP/P indicates this instruction takes three cycles to refill the instruction queue if the branch is taken and one program fetch cycle if the branch is not taken.

Branch				Complementary Branch			
Test	Mnemonic	Opcode	Boolean	Test	Mnemonic	Opcode	Comment
$r > m$	BGT	2E	$Z + (N \oplus V) = 0$	$r \leq m$	BLE	2F	Signed
$r \geq m$	BGE	2C	$N \oplus V = 0$	$r < m$	BLT	2D	Signed
$r = m$	BEQ	27	$Z = 1$	$r \neq m$	BNE	26	Signed
$r \leq m$	BLE	2F	$Z + (N \oplus V) = 1$	$r > m$	BGT	2E	Signed
$r < m$	BLT	2D	$N \oplus V = 1$	$r \geq m$	BGE	2C	Signed
$r > m$	BHI	22	$C + Z = 0$	$r \leq m$	BLS	23	Unsigned
$r \geq m$	BHS/BCC	24	$C = 0$	$r < m$	BLO/BCS	25	Unsigned
$r = m$	BEQ	27	$Z = 1$	$r \neq m$	BNE	26	Unsigned
$r \leq m$	BLS	23	$C + Z = 1$	$r > m$	BHI	22	Unsigned
$r < m$	BLO/BCS	25	$C = 1$	$r \geq m$	BHS/BCC	24	Unsigned
Carry	BCS	25	$C = 1$	No Carry	BCC	24	Simple
Negative	BMI	2B	$N = 1$	Plus	BPL	2A	Simple
Overflow	BVS	29	$V = 1$	No Overflow	BVC	28	Simple
$r = 0$	BEQ	27	$Z = 1$	$r \neq 0$	BNE	26	Simple
Always	BRA	20	—	Never	BRN	21	Unconditional

# BCLR

Clear Bits in Memory

# BCLR

**Operation:**  $(M) \bullet (\overline{\text{Mask}}) \Rightarrow M$

**Description:** Clears bits in location M. To clear a bit, set the corresponding bit in the mask byte. Bits in M that correspond to zeros in the mask byte are not changed. Mask bytes can be located at PC + 2, PC + 3, or PC + 4, depending on addressing mode used.

**Condition Codes and Boolean Formulas:**

S	X	H	I	N	Z	V	C
—	—	—	—	$\Delta$	$\Delta$	0	—

N: Set if MSB of result is set; cleared otherwise.

Z: Set if result is \$00; cleared otherwise.

V: 0; Cleared.

**Addressing Modes, Machine Code, and Execution Times:**

Source Form	Address Mode <sup>1</sup>	Object Code	Cycles	Access Detail
BCLR <i>opr8a, msk8</i>	DIR	4D dd mm	4	rPOw
BCLR <i>opr16a, msk8</i>	EXT	1D hh ll mm	4	rPPw
BCLR <i>opr0_xysp, msk8</i>	IDX	0D xb mm	4	rPOw
BCLR <i>opr9,xysp, msk8</i>	IDX1	0D xb ff mm	4	rPwP
BCLR <i>opr16,xysp, msk8</i>	IDX2	0D xb ee ff mm	6	frPwOP

Notes:

1. Indirect forms of indexed addressing cannot be used with this instruction.

# BCS

Branch if Carry Set  
(Same as BLO)

# BCS

**Operation:** If C = 1, then (PC) + \$0002 + Rel  $\Rightarrow$  PC

Simple branch

**Description:** Tests the C status bit and branches if C = 1.

See **3.7 Relative Addressing Mode** for details of branch execution.

**Condition Codes and Boolean Formulas:**

S	X	H	I	N	Z	V	C
-	-	-	-	-	-	-	-

None affected.

**Addressing Modes, Machine Code, and Execution Times:**

Source Form	Address Mode	Object Code	Cycles	Access Detail
BCS <i>rel8</i>	REL	25 rr	3/1	PPP/P <sup>1</sup>

Notes:

1. PPP/P indicates this instruction takes three cycles to refill the instruction queue if the branch is taken and one program fetch cycle if the branch is not taken.

Branch				Complementary Branch			
Test	Mnemonic	Opcode	Boolean	Test	Mnemonic	Opcode	Comment
r>m	BGT	2E	$Z + (N \oplus V) = 0$	r≤m	BLE	2F	Signed
r≥m	BGE	2C	$N \oplus V = 0$	r<m	BLT	2D	Signed
r=m	BEQ	27	$Z = 1$	r≠m	BNE	26	Signed
r≤m	BLE	2F	$Z + (N \oplus V) = 1$	r>m	BGT	2E	Signed
r<m	BLT	2D	$N \oplus V = 1$	r≥m	BGE	2C	Signed
r>m	BHI	22	$C + Z = 0$	r≤m	BLS	23	Unsigned
r≥m	BHS/BCC	24	$C = 0$	r<m	BLO/BCS	25	Unsigned
r=m	BEQ	27	$Z = 1$	r≠m	BNE	26	Unsigned
r≤m	BLS	23	$C + Z = 1$	r>m	BHI	22	Unsigned
r<m	BLO/BCS	25	$C = 1$	r≥m	BHS/BCC	24	Unsigned
Carry	BCS	25	$C = 1$	No Carry	BCC	24	Simple
Negative	BMI	2B	$N = 1$	Plus	BPL	2A	Simple
Overflow	BVS	29	$V = 1$	No Overflow	BVC	28	Simple
r=0	BEQ	27	$Z = 1$	r≠0	BNE	26	Simple
Always	BRA	20	—	Never	BRN	21	Unconditional

# BEQ

## Branch if Equal

# BEQ

**Operation:** If  $Z = 1$ , then  $(PC) + \$0002 + \text{Rel} \Rightarrow PC$

Simple branch

**Description:** Tests the Z status bit and branches if  $Z = 1$ .

See **3.7 Relative Addressing Mode** for details of branch execution.

### Condition Codes and Boolean Formulas:

S	X	H	I	N	Z	V	C
—	—	—	—	—	—	—	—

None affected.

### Addressing Modes, Machine Code, and Execution Times:

Source Form	Address Mode	Object Code	Cycles	Access Detail
BEQ <i>rel8</i>	REL	27 <i>rr</i>	3/1	PPP/P <sup>1</sup>

Notes:

1. PPP/P indicates this instruction takes three cycles to refill the instruction queue if the branch is taken and one program fetch cycle if the branch is not taken.

Branch				Complementary Branch			
Test	Mnemonic	Opcode	Boolean	Test	Mnemonic	Opcode	Comment
$r > m$	BGT	2E	$Z + (N \oplus V) = 0$	$r \leq m$	BLE	2F	Signed
$r \geq m$	BGE	2C	$N \oplus V = 0$	$r < m$	BLT	2D	Signed
$r = m$	BEQ	27	$Z = 1$	$r \neq m$	BNE	26	Signed
$r \leq m$	BLE	2F	$Z + (N \oplus V) = 1$	$r > m$	BGT	2E	Signed
$r < m$	BLT	2D	$N \oplus V = 1$	$r \geq m$	BGE	2C	Signed
$r > m$	BHI	22	$C + Z = 0$	$r \leq m$	BLS	23	Unsigned
$r \geq m$	BHS/BCC	24	$C = 0$	$r < m$	BLO/BCS	25	Unsigned
$r = m$	BEQ	27	$Z = 1$	$r \neq m$	BNE	26	Unsigned
$r \leq m$	BLS	23	$C + Z = 1$	$r > m$	BHI	22	Unsigned
$r < m$	BLO/BCS	25	$C = 1$	$r \geq m$	BHS/BCC	24	Unsigned
Carry	BCS	25	$C = 1$	No Carry	BCC	24	Simple
Negative	BMI	2B	$N = 1$	Plus	BPL	2A	Simple
Overflow	BVS	29	$V = 1$	No Overflow	BVC	28	Simple
$r = 0$	BEQ	27	$Z = 1$	$r \neq 0$	BNE	26	Simple
Always	BRA	20	—	Never	BRN	21	Unconditional

# BGE

Branch if Greater than or Equal to Zero

# BGE

**Operation:** If  $N \oplus V = 0$ , then  $(PC) + \$0002 + Rel \Rightarrow PC$

For signed two's complement values  
if (Accumulator)  $\geq$  (Memory), then branch

**Description:** If BGE is executed immediately after execution of CBA, CMPA, CMPB, CMPD, CPX, CPY, SBA, SUBA, SUBB, or SUBD, a branch occurs if and only if the signed two's complement number in the accumulator is greater than or equal to the signed two's complement number in memory.

See **3.7 Relative Addressing Mode** for details of branch execution.

### Condition Codes and Boolean Formulas:

S	X	H	I	N	Z	V	C
-	-	-	-	-	-	-	-

None affected.

### Addressing Modes, Machine Code, and Execution Times:

Source Form	Address Mode	Object Code	Cycles	Access Detail
BGE <i>relB</i>	REL	2C rr	3/1	PPP/P <sup>1</sup>

Notes:

1. PPP/P indicates this instruction takes three cycles to refill the instruction queue if the branch is taken and one program fetch cycle if the branch is not taken.

Branch				Complementary Branch			
Test	Mnemonic	Opcode	Boolean	Test	Mnemonic	Opcode	Comment
$r > m$	BGT	2E	$Z + (N \oplus V) = 0$	$r \leq m$	BLE	2F	Signed
$r \geq m$	BGE	2C	$N \oplus V = 0$	$r < m$	BLT	2D	Signed
$r = m$	BEQ	27	$Z = 1$	$r \neq m$	BNE	26	Signed
$r \leq m$	BLE	2F	$Z + (N \oplus V) = 1$	$r > m$	BGT	2E	Signed
$r < m$	BLT	2D	$N \oplus V = 1$	$r \geq m$	BGE	2C	Signed
$r > m$	BHI	22	$C + Z = 0$	$r \leq m$	BLS	23	Unsigned
$r \geq m$	BHS/BCC	24	$C = 0$	$r < m$	BLO/BCS	25	Unsigned
$r = m$	BEQ	27	$Z = 1$	$r \neq m$	BNE	26	Unsigned
$r \leq m$	BLS	23	$C + Z = 1$	$r > m$	BHI	22	Unsigned
$r < m$	BLO/BCS	25	$C = 1$	$r \geq m$	BHS/BCC	24	Unsigned
Carry	BCS	25	$C = 1$	No Carry	BCC	24	Simple
Negative	BMI	2B	$N = 1$	Plus	BPL	2A	Simple
Overflow	BVS	29	$V = 1$	No Overflow	BVC	28	Simple
$r = 0$	BEQ	27	$Z = 1$	$r \neq 0$	BNE	26	Simple
Always	BRA	20	—	Never	BRN	21	Unconditional

# BGND

## Enter Background Debug Mode

# BGND

**Description:** BGND operates like a software interrupt, except that no registers are stacked. First, the current PC value is stored in internal CPU register TMP2. Next, the BDM ROM and background register block become active. The BDM ROM contains a substitute vector, mapped to the address of the software interrupt vector, which points to routines in the BDM ROM that control background operation. The substitute vector is fetched, and execution continues from the address that it points to. Finally, the CPU checks the location that TMP2 points to. If the value stored in that location is \$00 (the BGND opcode), TMP2 is incremented, so that the instruction that follows the BGND instruction is the first instruction executed when normal program execution resumes.

For all other types of BDM entry, the CPU performs the same sequence of operations as for a BGND instruction, but the value stored in TMP2 already points to the instruction that would have executed next had BDM not become active. If active BDM is triggered just as a BGND instruction is about to execute, the BDM firmware does increment TMP2, but the change does not affect resumption of normal execution.

While BDM is active, the CPU executes debugging commands received via a special single-wire serial interface. BDM is terminated by the execution of specific debugging commands. Upon exit from BDM, the background/boot ROM and registers are disabled, the instruction queue is refilled starting with the return address pointed to by TMP2, and normal processing resumes.

BDM is normally disabled to avoid accidental entry. While BDM is disabled, BGND executes as described, but the firmware causes execution to return to the user program. Refer to **SECTION 8 DEVELOPMENT AND DEBUG SUPPORT** for more information concerning BDM.

### Condition Codes and Boolean Formulas:

S	X	H	I	N	Z	V	C
-	-	-	-	-	-	-	-

None affected.

### Addressing Modes, Machine Code, and Execution Times:

Source Form	Address Mode	Object Code	Cycles	Access Detail
BGND	INH	00	5	VfPPP

# BGT

## Branch if Greater than Zero

# BGT

**Operation:** If  $Z + (N \oplus V) = 0$ , then  $(PC) + \$0002 + Rel \Rightarrow PC$

For signed two's complement values  
if (Accumulator) > (Memory), then branch

**Description:** If BGT is executed immediately after execution of CBA, CMPA, CMPB, CMPD, CPX, CPY, SBA, SUBA, SUBB, or SUBD, a branch occurs if and only if the signed two's complement number in the accumulator is greater than the signed two's complement number in memory.

See 3.7 Relative Addressing Mode for details of branch execution.

### Condition Codes and Boolean Formulas:

S	X	H	I	N	Z	V	C
-	-	-	-	-	-	-	-

None affected.

### Addressing Modes, Machine Code, and Execution Times:

Source Form	Address Mode	Object Code	Cycles	Access Detail
BGT <i>relB</i>	REL	2E rr	3/1	PPP/P <sup>1</sup>

Notes:

1. PPP/P indicates this instruction takes three cycles to refill the instruction queue if the branch is taken and one program fetch cycle if the branch is not taken.

Branch				Complementary Branch			
Test	Mnemonic	Opcode	Boolean	Test	Mnemonic	Opcode	Comment
$r > m$	BGT	2E	$Z + (N \oplus V) = 0$	$r \leq m$	BLE	2F	Signed
$r \geq m$	BGE	2C	$N \oplus V = 0$	$r < m$	BLT	2D	Signed
$r = m$	BEQ	27	$Z = 1$	$r \neq m$	BNE	26	Signed
$r \leq m$	BLE	2F	$Z + (N \oplus V) = 1$	$r > m$	BGT	2E	Signed
$r < m$	BLT	2D	$N \oplus V = 1$	$r \geq m$	BGE	2C	Signed
$r > m$	BHI	22	$C + Z = 0$	$r \leq m$	BLS	23	Unsigned
$r \geq m$	BHS/BCC	24	$C = 0$	$r < m$	BLO/BCS	25	Unsigned
$r = m$	BEQ	27	$Z = 1$	$r \neq m$	BNE	26	Unsigned
$r \leq m$	BLS	23	$C + Z = 1$	$r > m$	BHI	22	Unsigned
$r < m$	BLO/BCS	25	$C = 1$	$r \geq m$	BHS/BCC	24	Unsigned
Carry	BCS	25	$C = 1$	No Carry	BCC	24	Simple
Negative	BMI	2B	$N = 1$	Plus	BPL	2A	Simple
Overflow	BVS	29	$V = 1$	No Overflow	BVC	28	Simple
$r = 0$	BEQ	27	$Z = 1$	$r \neq 0$	BNE	26	Simple
Always	BRA	20	—	Never	BRN	21	Unconditional

# BHI

## Branch if Higher

# BHI

**Operation:** If  $C + Z = 0$ , then  $(PC) + \$0002 + \text{Rel} \Rightarrow PC$

For unsigned values, if (Accumulator) > (Memory), then branch

**Description:** If BHI is executed immediately after execution of CBA, CMPA, CMPB, CMPD, CPX, CPY, SBA, SUBA, SUBB, or SUBD, a branch occurs if and only if the unsigned binary number in the accumulator was greater than the unsigned binary number in memory. Generally not useful after INC/DEC, LD/ST, TST/CLR/COM because these instructions do not affect the C status bit.

See **3.7 Relative Addressing Mode** for details of branch execution.

### Condition Codes and Boolean Formulas:

S	X	H	I	N	Z	V	C
—	—	—	—	—	—	—	—

None affected.

### Addressing Modes, Machine Code, and Execution Times:

Source Form	Address Mode	Object Code	Cycles	Access Detail
BHI <i>relB</i>	REL	22 rr	3/1	PPP/P <sup>1</sup>

Notes:

1. PPP/P indicates this instruction takes three cycles to refill the instruction queue if the branch is taken and one program fetch cycle if the branch is not taken.

Branch				Complementary Branch			
Test	Mnemonic	Opcode	Boolean	Test	Mnemonic	Opcode	Comment
$r > m$	BGT	2E	$Z + (N \oplus V) = 0$	$r \leq m$	BLE	2F	Signed
$r \geq m$	BGE	2C	$N \oplus V = 0$	$r < m$	BLT	2D	Signed
$r = m$	BEQ	27	$Z = 1$	$r \neq m$	BNE	26	Signed
$r \leq m$	BLE	2F	$Z + (N \oplus V) = 1$	$r > m$	BGT	2E	Signed
$r < m$	BLT	2D	$N \oplus V = 1$	$r \geq m$	BGE	2C	Signed
$r > m$	BHI	22	$C + Z = 0$	$r \leq m$	BLS	23	Unsigned
$r \geq m$	BHS/BCC	24	$C = 0$	$r < m$	BLO/BCS	25	Unsigned
$r = m$	BEQ	27	$Z = 1$	$r \neq m$	BNE	26	Unsigned
$r \leq m$	BLS	23	$C + Z = 1$	$r > m$	BHI	22	Unsigned
$r < m$	BLO/BCS	25	$C = 1$	$r \geq m$	BHS/BCC	24	Unsigned
Carry	BCS	25	$C = 1$	No Carry	BCC	24	Simple
Negative	BMI	2B	$N = 1$	Plus	BPL	2A	Simple
Overflow	BVS	29	$V = 1$	No Overflow	BVC	28	Simple
$r = 0$	BEQ	27	$Z = 1$	$r \neq 0$	BNE	26	Simple
Always	BRA	20	—	Never	BRN	21	Unconditional



# BHS

Branch if Higher or Same  
(Same as BCC)

# BHS

**Operation:** If  $C = 0$ , then  $(PC) + \$0002 + Rel \Rightarrow PC$

For unsigned values, if  $(\text{Accumulator}) \geq (\text{Memory})$ , then branch

**Description:** If BHS is executed immediately after execution of CBA, CMPA, CMPB, CMPD, CPX, CPY, SBA, SUBA, SUBB, or SUBD, a branch occurs if and only if the unsigned binary number in the accumulator was greater than the unsigned binary number in memory. Generally not useful after INC/DEC, LD/ST, TST/CLR/COM because these instructions do not affect the C status bit.

See 3.7 Relative Addressing Mode for details of branch execution.

**Condition Codes and Boolean Formulas:**

S	X	H	I	N	Z	V	C
-	-	-	-	-	-	-	-

None affected.

**Addressing Modes, Machine Code, and Execution Times:**

Source Form	Address Mode	Object Code	Cycles	Access Detail
BHS <i>relB</i>	REL	24 rr	3/1	PPP/P <sup>1</sup>

Notes:

1. PPP/P indicates this instruction takes three cycles to refill the instruction queue if the branch is taken and one program fetch cycle if the branch is not taken.

Branch				Complementary Branch			
Test	Mnemonic	Opcode	Boolean	Test	Mnemonic	Opcode	Comment
$r > m$	BGT	2E	$Z + (N \oplus V) = 0$	$r \leq m$	BLE	2F	Signed
$r \geq m$	BGE	2C	$N \oplus V = 0$	$r < m$	BLT	2D	Signed
$r = m$	BEQ	27	$Z = 1$	$r \neq m$	BNE	26	Signed
$r \leq m$	BLE	2F	$Z + (N \oplus V) = 1$	$r > m$	BGT	2E	Signed
$r < m$	BLT	2D	$N \oplus V = 1$	$r \geq m$	BGE	2C	Signed
$r > m$	BHI	22	$C + Z = 0$	$r \leq m$	BLS	23	Unsigned
$r \geq m$	BHS/BCC	24	$C = 0$	$r < m$	BLO/BCS	25	Unsigned
$r = m$	BEQ	27	$Z = 1$	$r \neq m$	BNE	26	Unsigned
$r \leq m$	BLS	23	$C + Z = 1$	$r > m$	BHI	22	Unsigned
$r < m$	BLO/BCS	25	$C = 1$	$r \geq m$	BHS/BCC	24	Unsigned
Carry	BCS	25	$C = 1$	No Carry	BCC	24	Simple
Negative	BMI	2B	$N = 1$	Plus	BPL	2A	Simple
Overflow	BVS	29	$V = 1$	No Overflow	BVC	28	Simple
$r = 0$	BEQ	27	$Z = 1$	$r \neq 0$	BNE	26	Simple
Always	BRA	20	—	Never	BRN	21	Unconditional

# BITA

## Bit Test A

# BITA

Operation: (A) • (M)

**Description:** Performs bitwise logical AND on the content of accumulator A and the content of memory location M, and modifies the condition codes accordingly. Each bit of the result is the logical AND of the corresponding bits of the accumulator and the memory location. Neither the content of the accumulator nor the content of the memory location is affected.

**Condition Codes and Boolean Formulas:**

S	X	H	I	N	Z	V	C
—	—	—	—	Δ	Δ	0	—

- N: Set if MSB of result is set; cleared otherwise.  
Z: Set if result is \$00; cleared otherwise.  
V: 0; Cleared.

**Addressing Modes, Machine Code, and Execution Times:**

Source Form	Address Mode	Object Code	Cycles	Access Detail
BITA #opr8i	IMM	85 ii	1	P
BITA opr8a	DIR	95 dd	3	rfrP
BITA opr16a	EXT	B5 hh ll	3	rOP
BITA oprx0_xysp	IDX	A5 xb	3	rfrP
BITA oprx9,xysp	IDX1	A5 xb ff	3	rPO
BITA oprx16,xysp	IDX2	A5 xb ee ff	4	frPP
BITA [D,xysp]	[D,IDX]	A5 xb	6	fIfrrP
BITA [oprx16,xysp]	[IDX2]	A5 xb ee ff	6	fIPrrP

# BITB

## Bit Test B

# BITB

Operation: (B) • (M)

**Description:** Performs bitwise logical AND on the content of accumulator B and the content of memory location M, and modifies the condition codes accordingly. Each bit of the result is the logical AND of the corresponding bits of the accumulator and the memory location. Neither the content of the accumulator nor the content of the memory location is affected.

**Condition Codes and Boolean Formulas:**

S	X	H	I	N	Z	V	C
-	-	-	-	Δ	Δ	0	-

- N: Set if MSB of result is set; cleared otherwise.  
Z: Set if result is \$00; cleared otherwise.  
V: 0; Cleared.

**Addressing Modes, Machine Code, and Execution Times:**

Source Form	Address Mode	Object Code	Cycles	Access Detail
BITB #opr8i	IMM	C5 ii	1	P
BITB opr8a	DIR	D5 dd	3	rfP
BITB opr16a	EXT	F5 hh ll	3	rOP
BITB oprx0_xysp	IDX	E5 xb	3	rfP
BITB oprx9_xysp	IDX1	E5 xb ff	3	rPO
BITB oprx16_xysp	IDX2	E5 xb ee ff	4	frPP
BITB [D,xysp]	[D,IDX]	E5 xb	6	fiPrfP
BITB [opr16,xysp]	[IDX2]	E5 xb ee ff	6	fiPrfP

# BLE

## Branch if Less Than or Equal to Zero

# BLE

**Operation:** If  $Z + (N \oplus V) = 1$ , then  $(PC) + \$0002 + Rel \Rightarrow PC$

For signed two's complement numbers  
if  $(\text{Accumulator}) \leq (\text{Memory})$ , then branch

**Description:** If BLE is executed immediately after execution of CBA, CMPA, CMPB, CMPD, CPX, CPY, SBA, SUBA, SUBB, or SUBD, a branch occurs if and only if the two's complement number in the accumulator was less than or equal to the two's complement number in memory.

See 3.7 Relative Addressing Mode for details of branch execution.

### Condition Codes and Boolean Formulas:

S	X	H	I	N	Z	V	C
-	-	-	-	-	-	-	-

None affected.

### Addressing Modes, Machine Code, and Execution Times:

Source Form	Address Mode	Object Code	Cycles	Access Detail
BLE <i>rel8</i>	REL	2F <i>rr</i>	3/1	PPP/P1

Notes:

1. PPP/P indicates this instruction takes three cycles to refill the instruction queue if the branch is taken and one program fetch cycle if the branch is not taken.

Branch				Complementary Branch			
Test	Mnemonic	Opcode	Boolean	Test	Mnemonic	Opcode	Comment
$r > m$	BGT	2E	$Z + (N \oplus V) = 0$	$r \leq m$	BLE	2F	Signed
$r \geq m$	BGE	2C	$N \oplus V = 0$	$r < m$	BLT	2D	Signed
$r = m$	BEQ	27	$Z = 1$	$r \neq m$	BNE	26	Signed
$r \leq m$	BLE	2F	$Z + (N \oplus V) = 1$	$r > m$	BGT	2E	Signed
$r < m$	BLT	2D	$N \oplus V = 1$	$r \geq m$	BGE	2C	Signed
$r > m$	BHI	22	$C + Z = 0$	$r \leq m$	BLS	23	Unsigned
$r \geq m$	BHS/BCC	24	$C = 0$	$r < m$	BLO/BCS	25	Unsigned
$r = m$	BEQ	27	$Z = 1$	$r \neq m$	BNE	26	Unsigned
$r \leq m$	BLS	23	$C + Z = 1$	$r > m$	BHI	22	Unsigned
$r < m$	BLO/BCS	25	$C = 1$	$r \geq m$	BHS/BCC	24	Unsigned
Carry	BCS	25	$C = 1$	No Carry	BCC	24	Simple
Negative	BMI	2B	$N = 1$	Plus	BPL	2A	Simple
Overflow	BVS	29	$V = 1$	No Overflow	BVC	28	Simple
$r = 0$	BEQ	27	$Z = 1$	$r \neq 0$	BNE	26	Simple
Always	BRA	20	—	Never	BRN	21	Unconditional

# BLO

Branch if Lower  
(Same as BCS)

# BLO

**Operation:** If  $C = 1$ , then  $(PC) + \$0002 + Rel \Rightarrow PC$

For unsigned values, if (Accumulator) < (Memory), then branch

**Description:** If BLO is executed immediately after execution of CBA, CMPA, CMPB, CMPD, CPX, CPY, SBA, SUBA, SUBB, or SUBD, a branch occurs if and only if the unsigned binary number in the accumulator is less than the unsigned binary number in memory. Generally not useful after INC/DEC, LD/ST, TST/CLR/COM because these instructions do not affect the C status bit.

See 3.7 Relative Addressing Mode for details of branch execution.

## Condition Codes and Boolean Formulas:

S	X	H	I	N	Z	V	C
-	-	-	-	-	-	-	-

None affected.

## Addressing Modes, Machine Code, and Execution Times:

Source Form	Address Mode	Object Code	Cycles	Access Detail
BLO <i>relB</i>	REL	25 <i>rr</i>	3/1	PPP/P <sup>1</sup>

Notes:

1. PPP/P indicates this instruction takes three cycles to refill the instruction queue if the branch is taken and one program fetch cycle if the branch is not taken.

Branch				Complementary Branch			
Test	Mnemonic	Opcode	Boolean	Test	Mnemonic	Opcode	Comment
$r > m$	BGT	2E	$Z + (N \oplus V) = 0$	$r \leq m$	BLE	2F	Signed
$r \geq m$	BGE	2C	$N \oplus V = 0$	$r < m$	BLT	2D	Signed
$r = m$	BEQ	27	$Z = 1$	$r \neq m$	BNE	26	Signed
$r \leq m$	BLE	2F	$Z + (N \oplus V) = 1$	$r > m$	BGT	2E	Signed
$r < m$	BLT	2D	$N \oplus V = 1$	$r \geq m$	BGE	2C	Signed
$r > m$	BHI	22	$C + Z = 0$	$r \leq m$	BLS	23	Unsigned
$r \geq m$	BHS/BCC	24	$C = 0$	$r < m$	BLO/BCS	25	Unsigned
$r = m$	BEQ	27	$Z = 1$	$r \neq m$	BNE	26	Unsigned
$r \leq m$	BLS	23	$C + Z = 1$	$r > m$	BHI	22	Unsigned
$r < m$	BLO/BCS	25	$C = 1$	$r \geq m$	BHS/BCC	24	Unsigned
Carry	BCS	25	$C = 1$	No Carry	BCC	24	Simple
Negative	BMI	2B	$N = 1$	Plus	BPL	2A	Simple
Overflow	BVS	29	$V = 1$	No Overflow	BVC	28	Simple
$r = 0$	BEQ	27	$Z = 1$	$r \neq 0$	BNE	26	Simple
Always	BRA	20	—	Never	BRN	21	Unconditional

# BLS

## Branch if Lower or Same

# BLS

**Operation:** If  $C + Z = 1$ , then  $(PC) + \$0002 + Rel \Rightarrow PC$

For unsigned values, if  $(\text{Accumulator}) \leq (\text{Memory})$ , then branch

**Description:** If BLS is executed immediately after execution of CBA, CMPA, CMPB, CMPD, CPX, CPY, SBA, SUBA, SUBB, or SUBD, a branch occurs if and only if the unsigned binary number in the accumulator is less than or equal to the unsigned binary number in memory. Generally not useful after INC/DEC, LD/ST, TST/CLR/COM because these instructions do not affect the C status bit.

See **3.7 Relative Addressing Mode** for details of branch execution.

### Condition Codes and Boolean Formulas:

S	X	H	I	N	Z	V	C
—	—	—	—	—	—	—	—

None affected.

### Addressing Modes, Machine Code, and Execution Times:

Source Form	Address Mode	Object Code	Cycles	Access Detail
BLS <i>relB</i>	REL	23 rr	3/1	PPP/P <sup>1</sup>

Notes:

1. PPP/P indicates this instruction takes three cycles to refill the instruction queue if the branch is taken and one program fetch cycle if the branch is not taken.

Branch				Complementary Branch			
Test	Mnemonic	Opcode	Boolean	Test	Mnemonic	Opcode	Comment
$r > m$	BGT	2E	$Z + (N \oplus V) = 0$	$r \leq m$	BLE	2F	Signed
$r \geq m$	BGE	2C	$N \oplus V = 0$	$r < m$	BLT	2D	Signed
$r = m$	BEQ	27	$Z = 1$	$r \neq m$	BNE	26	Signed
$r \leq m$	BLE	2F	$Z + (N \oplus V) = 1$	$r > m$	BGT	2E	Signed
$r < m$	BLT	2D	$N \oplus V = 1$	$r \geq m$	BGE	2C	Signed
$r > m$	BHI	22	$C + Z = 0$	$r \leq m$	BLS	23	Unsigned
$r \geq m$	BHS/BCC	24	$C = 0$	$r < m$	BLO/BCS	25	Unsigned
$r = m$	BEQ	27	$Z = 1$	$r \neq m$	BNE	26	Unsigned
$r \leq m$	BLS	23	$C + Z = 1$	$r > m$	BHI	22	Unsigned
$r < m$	BLO/BCS	25	$C = 1$	$r \geq m$	BHS/BCC	24	Unsigned
Carry	BCS	25	$C = 1$	No Carry	BCC	24	Simple
Negative	BMI	2B	$N = 1$	Plus	BPL	2A	Simple
Overflow	BVS	29	$V = 1$	No Overflow	BVC	28	Simple
$r = 0$	BEQ	27	$Z = 1$	$r \neq 0$	BNE	26	Simple
Always	BRA	20	—	Never	BRN	21	Unconditional

# BLT

## Branch if Less than Zero

# BLT

**Operation:** If  $N \oplus V = 1$ , then  $(PC) + \$0002 + Rel \Rightarrow PC$

For signed two's complement numbers  
if (Accumulator) < (Memory), then branch

**Description:** If BLT is executed immediately after execution of CBA, CMPA, CMPB, CMPD, CPX, CPY, SBA, SUBA, SUBB, or SUBD, a branch occurs if and only if the two's complement number in the accumulator is less than the two's complement number in memory.

See 3.7 Relative Addressing Mode for details of branch execution.

### Condition Codes and Boolean Formulas:

S	X	H	I	N	Z	V	C
-	-	-	-	-	-	-	-

None affected.

### Addressing Modes, Machine Code, and Execution Times:

Source Form	Address Mode	Object Code	Cycles	Access Detail
BLT <i>rel8</i>	REL	2D <i>rr</i>	3/1	PPP/P1

Notes:

1. PPP/P indicates this instruction takes three cycles to refill the instruction queue if the branch is taken and one program fetch cycle if the branch is not taken.

Branch				Complementary Branch			
Test	Mnemonic	Opcode	Boolean	Test	Mnemonic	Opcode	Comment
$r > m$	BGT	2E	$Z + (N \oplus V) = 0$	$r \leq m$	BLE	2F	Signed
$r \geq m$	BGE	2C	$N \oplus V = 0$	$r < m$	BLT	2D	Signed
$r = m$	BEQ	27	$Z = 1$	$r \neq m$	BNE	26	Signed
$r \leq m$	BLE	2F	$Z + (N \oplus V) = 1$	$r > m$	BGT	2E	Signed
$r < m$	BLT	2D	$N \oplus V = 1$	$r \geq m$	BGE	2C	Signed
$r > m$	BHI	22	$C + Z = 0$	$r \leq m$	BLS	23	Unsigned
$r \geq m$	BHS/BCC	24	$C = 0$	$r < m$	BLO/BCS	25	Unsigned
$r = m$	BEQ	27	$Z = 1$	$r \neq m$	BNE	26	Unsigned
$r \leq m$	BLS	23	$C + Z = 1$	$r > m$	BHI	22	Unsigned
$r < m$	BLO/BCS	25	$C = 1$	$r \geq m$	BHS/BCC	24	Unsigned
Carry	BCS	25	$C = 1$	No Carry	BCC	24	Simple
Negative	BMI	2B	$N = 1$	Plus	BPL	2A	Simple
Overflow	BVS	29	$V = 1$	No Overflow	BVC	28	Simple
$r = 0$	BEQ	27	$Z = 1$	$r \neq 0$	BNE	26	Simple
Always	BRA	20	—	Never	BRN	21	Unconditional

# BMI

## Branch if Minus

# BMI

**Operation:** If N = 1, then (PC) + \$0002 + Rel  $\Rightarrow$  PC

Simple branch

**Description:** Tests the N status bit and branches if N = 1.

See 3.7 Relative Addressing Mode for details of branch execution.

### Condition Codes and Boolean Formulas:

S	X	H	I	N	Z	V	C
-	-	-	-	-	-	-	-

None affected.

### Addressing Modes, Machine Code, and Execution Times:

Source Form	Address Mode	Object Code	Cycles	Access Detail
BMI <i>relB</i>	REL	2B <i>rr</i>	3/1	PPP/P <sup>1</sup>

Notes:

1. PPP/P indicates this instruction takes three cycles to refill the instruction queue if the branch is taken and one program fetch cycle if the branch is not taken.

Branch				Complementary Branch			
Test	Mnemonic	Opcode	Boolean	Test	Mnemonic	Opcode	Comment
$r > m$	BGT	2E	$Z + (N \oplus V) = 0$	$r \leq m$	BLE	2F	Signed
$r \geq m$	BGE	2C	$N \oplus V = 0$	$r < m$	BLT	2D	Signed
$r = m$	BEQ	27	$Z = 1$	$r \neq m$	BNE	26	Signed
$r \leq m$	BLE	2F	$Z + (N \oplus V) = 1$	$r > m$	BGT	2E	Signed
$r < m$	BLT	2D	$N \oplus V = 1$	$r \geq m$	BGE	2C	Signed
$r > m$	BHI	22	$C + Z = 0$	$r \leq m$	BLS	23	Unsigned
$r \geq m$	BHS/BCC	24	$C = 0$	$r < m$	BLO/BCS	25	Unsigned
$r = m$	BEQ	27	$Z = 1$	$r \neq m$	BNE	26	Unsigned
$r \leq m$	BLS	23	$C + Z = 1$	$r > m$	BHI	22	Unsigned
$r < m$	BLO/BCS	25	$C = 1$	$r \geq m$	BHS/BCC	24	Unsigned
Carry	BCS	25	$C = 1$	No Carry	BCC	24	Simple
Negative	BMI	2B	$N = 1$	Plus	BPL	2A	Simple
Overflow	BVS	29	$V = 1$	No Overflow	BVC	28	Simple
$r = 0$	BEQ	27	$Z = 1$	$r \neq 0$	BNE	26	Simple
Always	BRA	20	—	Never	BRN	21	Unconditional



# BNE

Branch if Not Equal to Zero

# BNE

**Operation:** If  $Z = 0$ , then  $(PC) + \$0002 + Rel \Rightarrow PC$

Simple branch

**Description:** Tests the Z status bit and branches if  $Z = 0$ .

See **3.7 Relative Addressing Mode** for details of branch execution.

**Condition Codes and Boolean Formulas:**

S	X	H	I	N	Z	V	C
—	—	—	—	—	—	—	—

None affected.

**Addressing Modes, Machine Code, and Execution Times:**

Source Form	Address Mode	Object Code	Cycles	Access Detail
BNE <i>relB</i>	REL	26 rr	3/1	PPP/P <sup>1</sup>

Notes:

1. PPP/P indicates this instruction takes three cycles to refill the instruction queue if the branch is taken and one program fetch cycle if the branch is not taken.

Branch				Complementary Branch			
Test	Mnemonic	Opcode	Boolean	Test	Mnemonic	Opcode	Comment
$r > m$	BGT	2E	$Z + (N \oplus V) = 0$	$r \leq m$	BLE	2F	Signed
$r \geq m$	BGE	2C	$N \oplus V = 0$	$r < m$	BLT	2D	Signed
$r = m$	BEQ	27	$Z = 1$	$r \neq m$	BNE	26	Signed
$r \leq m$	BLE	2F	$Z + (N \oplus V) = 1$	$r > m$	BGT	2E	Signed
$r < m$	BLT	2D	$N \oplus V = 1$	$r \geq m$	BGE	2C	Signed
$r > m$	BHI	22	$C + Z = 0$	$r \leq m$	BLS	23	Unsigned
$r \geq m$	BHS/BCC	24	$C = 0$	$r < m$	BLO/BCS	25	Unsigned
$r = m$	BEQ	27	$Z = 1$	$r \neq m$	BNE	26	Unsigned
$r \leq m$	BLS	23	$C + Z = 1$	$r > m$	BHI	22	Unsigned
$r < m$	BLO/BCS	25	$C = 1$	$r \geq m$	BHS/BCC	24	Unsigned
Carry	BCS	25	$C = 1$	No Carry	BCC	24	Simple
Negative	BMI	2B	$N = 1$	Plus	BPL	2A	Simple
Overflow	BVS	29	$V = 1$	No Overflow	BVC	28	Simple
$r = 0$	BEQ	27	$Z = 1$	$r \neq 0$	BNE	26	Simple
Always	BRA	20	—	Never	BRN	21	Unconditional

# BPL

## Branch if Plus

# BPL

**Operation:** If  $N = 0$ , then  $(PC) + \$0002 + Rel \Rightarrow PC$

Simple branch

**Description:** Tests the N status bit and branches if  $N = 0$ .

See **3.7 Relative Addressing Mode** for details of branch execution.

### Condition Codes and Boolean Formulas:

S	X	H	I	N	Z	V	C
—	—	—	—	—	—	—	—

None affected.

### Addressing Modes, Machine Code, and Execution Times:

Source Form	Address Mode	Object Code	Cycles	Access Detail
BPL <i>rel8</i>	REL	2A <i>rr</i>	3/1	PPP/P <sup>1</sup>

Notes:

1. PPP/P indicates this instruction takes three cycles to refill the instruction queue if the branch is taken and one program fetch cycle if the branch is not taken.

Branch				Complementary Branch			
Test	Mnemonic	Opcode	Boolean	Test	Mnemonic	Opcode	Comment
$r > m$	BGT	2E	$Z + (N \oplus V) = 0$	$r \leq m$	BLE	2F	Signed
$r \geq m$	BGE	2C	$N \oplus V = 0$	$r < m$	BLT	2D	Signed
$r = m$	BEQ	27	$Z = 1$	$r \neq m$	BNE	26	Signed
$r \leq m$	BLE	2F	$Z + (N \oplus V) = 1$	$r > m$	BGT	2E	Signed
$r < m$	BLT	2D	$N \oplus V = 1$	$r \geq m$	BGE	2C	Signed
$r > m$	BHI	22	$C + Z = 0$	$r \leq m$	BLS	23	Unsigned
$r \geq m$	BHS/BCC	24	$C = 0$	$r < m$	BLO/BCS	25	Unsigned
$r = m$	BEQ	27	$Z = 1$	$r \neq m$	BNE	26	Unsigned
$r \leq m$	BLS	23	$C + Z = 1$	$r > m$	BHI	22	Unsigned
$r < m$	BLO/BCS	25	$C = 1$	$r \geq m$	BHS/BCC	24	Unsigned
Carry	BCS	25	$C = 1$	No Carry	BCC	24	Simple
Negative	BMI	2B	$N = 1$	Plus	BPL	2A	Simple
Overflow	BVS	29	$V = 1$	No Overflow	BVC	28	Simple
$r = 0$	BEQ	27	$Z = 1$	$r \neq 0$	BNE	26	Simple
Always	BRA	20	—	Never	BRN	21	Unconditional

# BRA

Branch Always

# BRA

**Operation:** (PC) + \$0002 + Rel  $\Rightarrow$  PC

**Description:** Unconditional branch to an address calculated as shown in the expression. Rel is a relative offset stored as a two's complement number in the second byte of the branch instruction.

Execution time is longer when a conditional branch is taken than when it is not, because the instruction queue must be refilled before execution resumes at the new address. Since the BRA branch condition is always satisfied, the branch is always taken, and the instruction queue must always be refilled.

See **3.7 Relative Addressing Mode** for details of branch execution.

**Condition Codes and Boolean Formulas:**

S	X	H	I	N	Z	V	C
-	-	-	-	-	-	-	-

None affected.

**Addressing Modes, Machine Code, and Execution Times:**

Source Form	Address Mode	Object Code	Cycles	Access Detail
BRA rel8	REL	20 rr	3	PPP

# BRCLR

Branch if Bits Cleared

# BRCLR

**Operation:** If (M) • (Mask) = 0, then branch

**Description:** Performs bitwise logical AND on memory location M and the mask supplied with the instruction, then branches if and only if all bits with a value of one in the mask byte correspond to bits with a value of zero in the tested byte. Mask operands can be located at PC + 1, PC + 2, or PC + 4, depending on addressing mode. The branch offset is referenced to the next address after the relative offset (rr) which is the last byte of the instruction object code.

See **3.7 Relative Addressing Mode** for details of branch execution.

**Condition Codes and Boolean Formulas:**

S	X	H	I	N	Z	V	C
—	—	—	—	—	—	—	—

None affected.

**Addressing Modes, Machine Code, and Execution Times:**

Source Form	Address Mode <sup>1</sup>	Object Code	Cycles	Access Detail
BRCLR <i>opr8a, msk8, rel8</i>	DIR	4F dd mm rr	4	rPPP
BRCLR <i>opr16a, msk8, rel8</i>	EXT	1F hh ll mm rr	5	rfPPP
BRCLR <i>opr0_xysp, msk8, rel8</i>	IDX	0F xb mm rr	4	rPPP
BRCLR <i>opr9_xysp, msk8, rel8</i>	IDX1	0F xb ff mm rr	6	rffPPP
BRCLR <i>opr16_xysp, msk8, rel8</i>	IDX2	0F xb ee ff mm rr	8	frPffPPP

Notes:

1. Indirect forms of indexed addressing cannot be used with this instruction.

# BRN

Branch Never

# BRN

**Operation:** (PC) + \$0002 ⇒ PC

**Description:** Never branches. BRN is effectively a 2-byte NOP that requires one cycle to execute. BRN is included in the instruction set to provide a complement to the BRA instruction. The instruction is useful during program debug, to negate the effect of another branch instruction without disturbing the offset byte. A complement for BRA is also useful in compiler implementations.

Execution time is longer when a conditional branch is taken than when it is not, because the instruction queue must be refilled before execution resumes at the new address. Since the BRN branch condition is never satisfied, the branch is never taken, and only a single program fetch is needed to update the instruction queue.

See **3.7 Relative Addressing Mode** for details of branch execution.

**Condition Codes and Boolean Formulas:**

S	X	H	I	N	Z	V	C
-	-	-	-	-	-	-	-

None affected.

**Addressing Modes, Machine Code, and Execution Times:**

Source Form	Address Mode	Object Code	Cycles	Access Detail
BRN <i>rel</i> 8	REL	21 rr	1	P

# BRSET

Branch if Bits Set

# BRSET

**Operation:** If  $(\overline{M}) \bullet (\text{Mask}) = 0$ , then branch

**Description:** Performs bitwise logical AND on the inverse of memory location M and the mask supplied with the instruction, then branches if and only if all bits with a value of one in the mask byte correspond to bits with a value of one in the tested byte. Mask operands can be located at PC + 1, PC + 2, or PC + 4, depending on addressing mode. The branch offset is referenced to the next address after the relative offset (rr) which is the last byte of the instruction object code.

See 3.7 Relative Addressing Mode for details of branch execution.

**Condition Codes and Boolean Formulas:**

S	X	H	I	N	Z	V	C
—	—	—	—	—	—	—	—

None affected.

**Addressing Modes, Machine Code, and Execution Times:**

Source Form	Address Mode <sup>1</sup>	Object Code	Cycles	Access Detail
BRSET <i>opr8a, msk8, rel8</i>	DIR	4E dd mm rr	4	rPPP
BRSET <i>opr16a, msk8, rel8</i>	EXT	1E hh ll mm rr	5	rfPPP
BRSET <i>opr0_xysp, msk8, rel8</i>	IDX	0E xb mm rr	4	rPPP
BRSET <i>opr9_xysp, msk8, rel8</i>	IDX1	0E xb ff mm rr	6	rffPPP
BRSET <i>opr16_xysp, msk8, rel8</i>	IDX2	0E xb ee ff mm rr	8	frPfPPP

Notes:

1. Indirect forms of indexed addressing cannot be used with this instruction.

# BSET

## Set Bit(s) in Memory

# BSET

**Operation:** (M) + (Mask)  $\Rightarrow$  M

**Description:** Sets bits in memory location M. To set a bit, set the corresponding bit in the mask byte. All other bits in M are unchanged. The mask byte can be located at PC + 2, PC + 3, or PC + 4, depending upon addressing mode.

**Condition Codes and Boolean Formulas:**

S	X	H	I	N	Z	V	C
-	-	-	-	$\Delta$	$\Delta$	0	-

N: Set if MSB of result is set; cleared otherwise.

Z: Set if result is \$00; cleared otherwise.

V: 0; Cleared.

**Addressing Modes, Machine Code, and Execution Times:**

Source Form	Address Mode <sup>1</sup>	Object Code	Cycles	Access Detail
BSET <i>opr8a, msk8</i>	DIR	4C dd mm	4	rPOw
BSET <i>opr16a, msk8</i>	EXT	1C hh ll mm	4	rPPw
BSET <i>opr0_xysp, msk8</i>	IDX	0C xb mm	4	rPOw
BSET <i>opr9_xysp, msk8</i>	IDX1	0C xb ff mm	4	rPwP
BSET <i>opr16_xysp, msk8</i>	IDX2	0C xb ee ff mm	6	frPwOP

Notes:

1. Indirect forms of indexed addressing cannot be used with this instruction.

# BSR

## Branch to Subroutine

# BSR

**Operation:**  $(SP) - \$0002 \Rightarrow SP$   
 $RTN_H : RTN_L \Rightarrow M(SP) : M(SP + 1)$   
 $(PC) + Rel \Rightarrow PC$

**Description:** Sets up conditions to return to normal program flow, then transfers control to a subroutine. Uses the address of the instruction after the BSR as a return address.

Decrements the SP by two, to allow the two bytes of the return address to be stacked.

Stacks the return address (the SP points to the high order byte of the return address).

Branches to a location determined by the branch offset.

Subroutines are normally terminated with an RTS instruction, which restores the return address from the stack.

### Condition Codes and Boolean Formulas:

S	X	H	I	N	Z	V	C
-	-	-	-	-	-	-	-

None affected.

### Addressing Modes, Machine Code, and Execution Times:

Source Form	Address Mode	Object Code	Cycles	Access Detail
BSR <i>rel</i> 8	REL	07 <i>rr</i>	4	PPPS



# BVC

## Branch if Overflow Cleared

# BVC

**Operation:** If  $V = 0$ , then  $(PC) + \$0002 + Rel \Rightarrow PC$

Simple branch

**Description:** Tests the V status bit and branches if  $V = 0$ .

BVC causes a branch when a previous operation on two's complement binary values does not cause an overflow. That is, when BVC follows a two's complement operation, a branch occurs when the result of the operation is valid.

See **3.7 Relative Addressing Mode** for details of branch execution.

### Condition Codes and Boolean Formulas:

S	X	H	I	N	Z	V	C
-	-	-	-	-	-	-	-

None affected.

### Addressing Modes, Machine Code, and Execution Times:

Source Form	Address Mode	Object Code	Cycles	Access Detail
BVC <i>rel8</i>	REL	28 <i>rr</i>	3/1	PPP/P <sup>1</sup>

Notes:

1. PPP/P indicates this instruction takes three cycles to refill the instruction queue if the branch is taken and one program fetch cycle if the branch is not taken.

Branch				Complementary Branch			
Test	Mnemonic	Opcode	Boolean	Test	Mnemonic	Opcode	Comment
$r > m$	BGT	2E	$Z + (N \oplus V) = 0$	$r \leq m$	BLE	2F	Signed
$r \geq m$	BGE	2C	$N \oplus V = 0$	$r < m$	BLT	2D	Signed
$r = m$	BEQ	27	$Z = 1$	$r \neq m$	BNE	26	Signed
$r \leq m$	BLE	2F	$Z + (N \oplus V) = 1$	$r > m$	BGT	2E	Signed
$r < m$	BLT	2D	$N \oplus V = 1$	$r \geq m$	BGE	2C	Signed
$r > m$	BHI	22	$C + Z = 0$	$r \leq m$	BLS	23	Unsigned
$r \geq m$	BHS/BCC	24	$C = 0$	$r < m$	BLO/BCS	25	Unsigned
$r = m$	BEQ	27	$Z = 1$	$r \neq m$	BNE	26	Unsigned
$r \leq m$	BLS	23	$C + Z = 1$	$r > m$	BHI	22	Unsigned
$r < m$	BLO/BCS	25	$C = 1$	$r \geq m$	BHS/BCC	24	Unsigned
Carry	BCS	25	$C = 1$	No Carry	BCC	24	Simple
Negative	BMI	2B	$N = 1$	Plus	BPL	2A	Simple
Overflow	BVS	29	$V = 1$	No Overflow	BVC	28	Simple
$r = 0$	BEQ	27	$Z = 1$	$r \neq 0$	BNE	26	Simple
Always	BRA	20	—	Never	BRN	21	Unconditional

# BVS

## Branch if Overflow Set

# BVS

**Operation:** If  $V = 1$ , then  $(PC) + \$0002 + Rel \Rightarrow PC$

Simple branch

**Description:** Tests the V status bit and branches if  $V = 1$ .

BVS causes a branch when a previous operation on two's complement binary values causes an overflow. That is, when BVS follows a two's complement operation, a branch occurs when the result of the operation is invalid.

See **3.7 Relative Addressing Mode** for details of branch execution.

### Condition Codes and Boolean Formulas:

S	X	H	I	N	Z	V	C
—	—	—	—	—	—	—	—

None affected.

### Addressing Modes, Machine Code, and Execution Times:

Source Form	Address Mode	Object Code	Cycles	Access Detail
BVS <i>relB</i>	REL	29 rr	3/1	PPP/P <sup>1</sup>

Notes:

1. PPP/P indicates this instruction takes three cycles to refill the instruction queue if the branch is taken and one program fetch cycle if the branch is not taken.

Branch				Complementary Branch			
Test	Mnemonic	Opcode	Boolean	Test	Mnemonic	Opcode	Comment
$r > m$	BGT	2E	$Z + (N \oplus V) = 0$	$r \leq m$	BLE	2F	Signed
$r \geq m$	BGE	2C	$N \oplus V = 0$	$r < m$	BLT	2D	Signed
$r = m$	BEQ	27	$Z = 1$	$r \neq m$	BNE	26	Signed
$r \leq m$	BLE	2F	$Z + (N \oplus V) = 1$	$r > m$	BGT	2E	Signed
$r < m$	BLT	2D	$N \oplus V = 1$	$r \geq m$	BGE	2C	Signed
$r > m$	BHI	22	$C + Z = 0$	$r \leq m$	BLS	23	Unsigned
$r \geq m$	BHS/BCC	24	$C = 0$	$r < m$	BLO/BCS	25	Unsigned
$r = m$	BEQ	27	$Z = 1$	$r \neq m$	BNE	26	Unsigned
$r \leq m$	BLS	23	$C + Z = 1$	$r > m$	BHI	22	Unsigned
$r < m$	BLO/BCS	25	$C = 1$	$r \geq m$	BHS/BCC	24	Unsigned
Carry	BCS	25	$C = 1$	No Carry	BCC	24	Simple
Negative	BMI	2B	$N = 1$	Plus	BPL	2A	Simple
Overflow	BVS	29	$V = 1$	No Overflow	BVC	28	Simple
$r = 0$	BEQ	27	$Z = 1$	$r \neq 0$	BNE	26	Simple
Always	BRA	20	—	Never	BRN	21	Unconditional

# CALL

## Call Subroutine in Expanded Memory

# CALL

**Operation:** (SP) - \$0002  $\Rightarrow$  SP  
RTN<sub>H</sub> : RTN<sub>L</sub>  $\Rightarrow$  M(SP) : M(SP + 1)  
(SP) - \$0001  $\Rightarrow$  SP  
(PPAGE)  $\Rightarrow$  M(SP)  
page  $\Rightarrow$  PPAGE  
Subroutine Address  $\Rightarrow$  PC

**Description:** Sets up conditions to return to normal program flow, then transfers control to a subroutine in expanded memory. Uses the address of the instruction following the CALL as a return address. For code compatibility, CALL also executes correctly in devices that do not have expanded memory capability.

Decrements the SP by two, to allow the two bytes of the return address to be stacked.

Stacks the return address (the SP points to the high order byte of the return address).

Decrements the SP by one, to allow the current memory page value in the PPAGE register to be stacked.

Stacks the content of PPAGE.

Writes a new page value supplied by the instruction to PPAGE.

Transfers control to the subroutine.

In indexed-indirect modes, the subroutine address and the PPAGE value are fetched from memory in the order M high byte, M low byte, and new PPAGE value.

Expanded-memory subroutines must be terminated by an RTC instruction, which restores the return address and PPAGE value from the stack.

### Condition Codes and Boolean Formulas:

S	X	H	I	N	Z	V	C
-	-	-	-	-	-	-	-

None affected.

### Addressing Modes, Machine Code, and Execution Times:

Source Form	Address Mode	Object Code	Cycles	Access Detail
CALL <i>opr16a, page</i>	EXT	4A hh 11 pg	8	gnfSsPPP
CALL <i>opr0_xysp, page</i>	IDX	4B xb pg	8	gnfSsPPP
CALL <i>opr9,xysp, page</i>	IDX1	4B xb ff pg	8	gnfSsPPP
CALL <i>opr16,xysp, page</i>	IDX2	4B xb ee ff pg	9	fgnfSsPPP
CALL [D,xysp]	[D,IDX]	4B xb	10	fIignSsPPP
CALL [opr16,xysp]	[IDX2]	4B xb ee ff	10	fIignSsPPP

# CBA

## Compare Accumulators

# CBA

**Operation:** (A) – (B)

**Description:** Compares the content of accumulator A to the content of accumulator B and sets the condition codes, which may then be used for arithmetic and logical conditional branches. The contents of the accumulators are not changed.

### Condition Codes and Boolean Formulas:

S	X	H	I	N	Z	V	C
–	–	–	–	Δ	Δ	Δ	Δ

N: Set if MSB of result is set; cleared otherwise.

Z: Set if result is \$00; cleared otherwise.

V:  $A7 \bullet \overline{B7} \bullet \overline{R7} + \overline{A7} \bullet B7 \bullet R7$   
Set if a two's complement overflow resulted from the operation; cleared otherwise.

C:  $\overline{A7} \bullet B7 + B7 \bullet R7 + R7 + \overline{A7}$   
Set if there was a borrow from the MSB of the result; cleared otherwise.

### Addressing Modes, Machine Code, and Execution Times:

Source Form	Address Mode	Object Code	Cycles	Access Detail
CBA	INH	18 17	2	00

# CLC

Clear Carry

# CLC

**Operation:** 0 ⇒ C bit

**Description:** Clears the C status bit. This instruction is assembled as ANDCC #\$FE. The ANDCC instruction can be used to clear any combination of bits in the CCR in one operation.

CLC can be used to set up the C bit prior to a shift or rotate instruction involving the C bit.

**Condition Codes and Boolean Formulas:**

S	X	H	I	N	Z	V	C				
-	-	-	-	-	-	-	0				

C: 0; Cleared.

**Addressing Modes, Machine Code, and Execution Times:**

Source Form	Address Mode	Object Code	Cycles	Access Detail
CLC translates to... ANDCC #\$FE	IMM	10 FE	1	P

# CLI

## Clear Interrupt Mask

# CLI

**Operation:** 0  $\Rightarrow$  I bit

**Description:** Clears the I mask bit. This instruction is assembled as ANDCC #\$EF. The ANDCC instruction can be used to clear any combination of bits in the CCR in one operation.

When the I bit is cleared, interrupts are enabled. There is a one cycle (bus clock) delay in the clearing mechanism for the I bit so that, if interrupts were previously disabled, the next instruction after a CLI will always be executed, even if there was an interrupt pending prior to execution of the CLI instruction.

### Condition Codes and Boolean Formulas:

S	X	H	I	N	Z	V	C
-	-	-	0	-	-	-	-

I: 0; Cleared.

### Addressing Modes, Machine Code, and Execution Times:

Source Form	Address Mode	Object Code	Cycles	Access Detail
CLI translates to... ANDCC #\$EF	IMM	10 EF	1	P

# CLR

Clear Memory

# CLR

**Operation:**  $0 \Rightarrow M$

**Description:** All bits in memory location M are cleared to zero.

**Condition Codes and Boolean Formulas:**

S	X	H	I	N	Z	V	C
-	-	-	-	0	1	0	0

N: 0; Cleared.

Z: 1; Set.

V: 0; Cleared.

C: 0; Cleared.

**Addressing Modes, Machine Code, and Execution Times:**

Source Form	Address Mode	Object Code	Cycles	Access Detail
CLR <i>opr16a</i>	EXT	79 hh ll	3	wOP
CLR <i>opr0_xysp</i>	IDX	69 xb	2	Pw
CLR <i>opr9,xysp</i>	IDX1	69 xb ff	3	PwO
CLR <i>opr16,xysp</i>	IDX2	69 xb ee ff	3	PwP
CLR [D, <i>xysp</i> ]	[D,IDX]	69 xb	5	PIfPw
CLR [ <i>opr16,xysp</i> ]	[IDX2]	69 xb ee ff	5	PIPPw

# CLRA

Clear A

# CLRA

**Operation:**  $0 \Rightarrow A$

**Description:** All bits in accumulator A are cleared to zero.

**Condition Codes and Boolean Formulas:**

S	X	H	I	N	Z	V	C
—	—	—	—	0	1	0	0

N: 0; Cleared.

Z: 1; Set.

V: 0; Cleared.

C: 0; Cleared.

**Addressing Modes, Machine Code, and Execution Times:**

Source Form	Address Mode	Object Code	Cycles	Access Detail
CLRA	INH	87	1	0



# CLRB

Clear B

# CLRB

Operation:  $0 \Rightarrow B$

Description: All bits in accumulator B are cleared to zero.

Condition Codes and Boolean Formulas:

S	X	H	I	N	Z	V	C
-	-	-	-	0	1	0	0

- N: 0; Cleared.
- Z: 1; Set.
- V: 0; Cleared.
- C: 0; Cleared.

Addressing Modes, Machine Code, and Execution Times:

Source Form	Address Mode	Object Code	Cycles	Access Detail
CLRB	INH	C7	1	0

CLV

Clear Two's Complement Overflow Bit

CLV

**Operation:** 0 ⇒ V bit

**Description:** Clears the V status bit. This instruction is assembled as ANDCC #\$FD. The ANDCC instruction can be used to clear any combination of bits in the CCR in one operation.

**Condition Codes and Boolean Formulas:**

S	X	H	I	N	Z	V	C
-	-	-	-	-	-	0	-

V: 0; Cleared.

**Addressing Modes, Machine Code, and Execution Times:**

Source Form	Address Mode	Object Code	Cycles	Access Detail
CLV translates to... ANDCC #\$FD	IMM	10 FD	1	P

# CMPA

Compare A

# CMPA

**Operation:** (A) – (M)

**Description:** Compares the content of accumulator A to the content of memory location M and sets the condition codes, which may then be used for arithmetic and logical conditional branching. The contents of A and location M are not changed.

## Condition Codes and Boolean Formulas:

S	X	H	I	N	Z	V	C
–	–	–	–	Δ	Δ	Δ	Δ

N: Set if MSB of result is set; cleared otherwise.

Z: Set if result is \$00; cleared otherwise.

V:  $X7 \bullet \overline{M7} \bullet \overline{R7} + \overline{X7} \bullet M7 \bullet R7$

Set if a two's complement overflow resulted from the operation; cleared otherwise.

C:  $\overline{X7} \bullet M7 + M7 \bullet R7 + R7 + \overline{X7}$

Set if there was a borrow from the MSB of the result; cleared otherwise.

## Addressing Modes, Machine Code, and Execution Times:

Source Form	Address Mode	Object Code	Cycles	Access Detail
CMPA #opr8l	IMM	8l ii	1	P
CMPA opr8a	DIR	9l dd	3	rFP
CMPA opr16a	EXT	B1 hh ll	3	rOP
CMPA oprx0_xysp	IDX	A1 xb	3	rFP
CMPA oprx9_xysp	IDX1	A1 xb ff	3	rPO
CMPA oprx16_xysp	IDX2	A1 xb ee ff	4	frPP
CMPA [D,xysp]	[D,IDX]	A1 xb	6	fiFrFP
CMPA [oprx16,xysp]	[IDX2]	A1 xb ee ff	6	fiPrFP

# CMPB

Compare B

# CMPB

**Operation:** (B) – (M)

**Description:** Compares the content of accumulator B to the content of memory location M and sets the condition codes, which may then be used for arithmetic and logical conditional branching. The contents of B and location M are not changed.

## Condition Codes and Boolean Formulas:

S	X	H	I	N	Z	V	C
–	–	–	–	$\Delta$	$\Delta$	$\Delta$	$\Delta$

N: Set if MSB of result is set; cleared otherwise.

Z: Set if result is \$00; cleared otherwise.

V:  $X7 \bullet \overline{M7} \bullet \overline{R7} + \overline{X7} \bullet M7 \bullet R7$

Set if a two's complement overflow resulted from the operation; cleared otherwise.

C:  $\overline{X7} \bullet M7 + M7 \bullet R7 + R7 + \overline{X7}$

Set if there was a borrow from the MSB of the result; cleared otherwise.

## Addressing Modes, Machine Code, and Execution Times:

Source Form	Address Mode	Object Code	Cycles	Access Detail
CMPB #opr8i	IMM	C1 ii	1	P
CMPB opr8a	DIR	D1 dd	3	rfP
CMPB opr16a	EXT	F1 hh ll	3	rOP
CMPB oprx0_xysp	IDX	E1 xb	3	rfP
CMPB oprx9_xysp	IDX1	E1 xb ff	3	rPO
CMPB oprx16_xysp	IDX2	E1 xb ee ff	4	frPP
CMPB [D,xysp]	[D,IDX]	E1 xb	6	fIfrfP
CMPB [oprx16,xysp]	[IDX2]	E1 xb ee ff	6	fIPrfP

# COM

## Complement Memory

# COM

**Operation:**  $(\overline{M}) = \$FF - (M) \Rightarrow M$

**Description:** Replaces the content of memory location M with its one's complement. Each bit of M is complemented. Immediately after a COM operation on unsigned values, only the BEQ, BNE, LBEQ, and LBNE branches can be expected to perform consistently. After operation on two's complement values, all signed branches are available.

### Condition Codes and Boolean Formulas:

S	X	H	I	N	Z	V	C
-	-	-	-	$\Delta$	$\Delta$	0	1

N: Set if MSB of result is set; cleared otherwise.

Z: Set if result is \$00; cleared otherwise.

V: 0; Cleared.

C: 1; Set (for M6800 compatibility).

### Addressing Modes, Machine Code, and Execution Times:

Source Form	Address Mode	Object Code	Cycles	Access Detail
COM <i>opr16a</i>	EXT	71 hh ll	4	rOPw
COM <i>opr0_xysp</i>	IDX	61 xb	3	rPw
COM <i>opr9,xysp</i>	IDX1	61 xb ff	4	rPOw
COM <i>opr16,xysp</i>	IDX2	61 xb ee ff	5	frPPw
COM [D, <i>xysp</i> ]	[D,IDX]	61 xb	6	fIfrPw
COM [ <i>opr16,xysp</i> ]	[IDX2]	61 xb ee ff	6	fIPrPw

# COMA

Complement A

# COMA

**Operation:**  $(\bar{A}) = \$FF - (A) \Rightarrow A$

**Description:** Replaces the content of accumulator A with its one's complement. Each bit of A is complemented. Immediately after a COM operation on unsigned values, only the BEQ, BNE, LBEQ, and LBNE branches can be expected to perform consistently. After operation on two's complement values, all signed branches are available.

## Condition Codes and Boolean Formulas:

S	X	H	I	N	Z	V	C
-	-	-	-	$\Delta$	$\Delta$	0	1

N: Set if MSB of result is set; cleared otherwise.

Z: Set if result is \$00; cleared otherwise.

V: 0; Cleared.

C: 1; Set (for M6800 compatibility).

## Addressing Modes, Machine Code, and Execution Times:

Source Form	Address Mode	Object Code	Cycles	Access Detail
COMA	INH	41	1	0

# COMB

Complement B

# COMB

**Operation:**  $(\bar{B}) = \$FF - (B) \Rightarrow B$

**Description:** Replaces the content of accumulator B with its one's complement. Each bit of B is complemented. Immediately after a COM operation on unsigned values, only the BEQ, BNE, LBEQ, and LBNE branches can be expected to perform consistently. After operation on two's complement values, all signed branches are available.

**Condition Codes and Boolean Formulas:**

S	X	H	I	N	Z	V	C
-	-	-	-	$\Delta$	$\Delta$	0	1

N: Set if MSB of result is set; cleared otherwise.

Z: Set if result is \$00; cleared otherwise.

V: 0; Cleared.

C: 1; Set (for M6800 compatibility).

**Addressing Modes, Machine Code, and Execution Times:**

Source Form	Address Mode	Object Code	Cycles	Access Detail
COMB	INH	51	1	0

# CPD

## Compare Double Accumulator

# CPD

**Operation:**  $(A : B) - (M : M + 1)$

**Description:** Compares the content of double accumulator D with a 16-bit value at the address specified, and sets the condition codes accordingly. The compare is accomplished internally by a 16-bit subtract of  $(M : M + 1)$  from D without modifying either D or  $(M : M + 1)$ .

### Condition Codes and Boolean Formulas:

S	X	H	I	N	Z	V	C
—	—	—	—	$\Delta$	$\Delta$	$\Delta$	$\Delta$

N: Set if MSB of result is set; cleared otherwise.

Z: Set if result is \$0000; cleared otherwise.

V:  $D15 \bullet \overline{M15} \bullet \overline{R15} + D15 \bullet M15 \bullet R15$

Set if two's complement overflow resulted from the operation; cleared otherwise.

C:  $\overline{D15} \bullet M15 + M15 \bullet R15 + R15 + \overline{D15}$

Set if the absolute value of the content of memory is larger than the absolute value of the accumulator; cleared otherwise.

### Addressing Modes, Machine Code, and Execution Times:

Source Form	Address Mode	Object Code	Cycles	Access Detail
CPD #opr16i	IMM	8C jj kk	2	OP
CPD opr8a	DIR	9C dd	3	RfP
CPD opr16a	EXT	BC hh ll	3	ROP
CPD oprx0,xysp	IDX	AC xb	3	RfP
CPD oprx9,xysp	IDX1	AC xb ff	3	RPO
CPD oprx16,xysp	IDX2	AC xb ee ff	4	fRPP
CPD [D,xysp]	[D,IDX]	AC xb	6	fIfRfP
CPD [oprx16,xysp]	[IDX2]	AC xb ee ff	6	fIPRfP



# CPS

## Compare Stack Pointer

# CPS

**Operation:**  $(SP) - (M : M + 1)$

**Description:** Compares the content of the SP with a 16-bit value at the address specified, and sets the condition codes accordingly. The compare is accomplished internally by doing a 16-bit subtract of  $(M : M + 1)$  from the SP without modifying either the SP or  $(M : M + 1)$ .

**Condition Codes and Boolean Formulas:**

S	X	H	I	N	Z	V	C
-	-	-	-	$\Delta$	$\Delta$	$\Delta$	$\Delta$

N: Set if MSB of result is set; cleared otherwise.

Z: Set if result is \$0000; cleared otherwise.

V:  $S15 \bullet M15 \bullet \overline{R15} + \overline{S15} \bullet M15 \bullet R15$   
Set if two's complement overflow resulted from the operation; cleared otherwise.

C:  $\overline{S15} \bullet M15 + M15 \bullet R15 + R15 + \overline{S15}$   
Set if the absolute value of the content of memory is larger than the absolute value of the SP; cleared otherwise.

**Addressing Modes, Machine Code, and Execution Times:**

Source Form	Address Mode	Object Code	Cycles	Access Detail
CPS #opr16i	IMM	8F jj kk	2	OP
CPS opr8a	DIR	9F dd	3	RfP
CPS opr16a	EXT	BF hh ll	3	ROP
CPS oprx0,xysp	IDX	AF xb	3	RfP
CPS oprx9,xysp	IDX1	AF xb ff	3	RPO
CPS oprx16,xysp	IDX2	AF xb ee ff	4	fRPP
CPS [D,xysp]	[D,IDX]	AF xb	6	fIfRfP
CPS [oprx16,xysp]	[IDX2]	AF xb ee ff	6	fIPRfP

# CPX

## Compare Index Register X

# CPX

**Operation:**  $(X) - (M : M + 1)$

**Description:** Compares the content of index register X with a 16-bit value at the address specified, and sets the condition codes accordingly. The compare is accomplished internally by a 16-bit subtract of  $(M : M + 1)$  from index register X without modifying either index register X or  $(M : M + 1)$ .

### Condition Codes and Boolean Formulas:

S	X	H	I	N	Z	V	C
-	-	-	-	$\Delta$	$\Delta$	$\Delta$	$\Delta$

N: Set if MSB of result is set; cleared otherwise.

Z: Set if result is \$0000; cleared otherwise.

V:  $X15 \bullet \overline{M15} \bullet \overline{R15} + \overline{X15} \bullet M15 \bullet R15$   
Set if two's complement overflow resulted from the operation; cleared otherwise.

C:  $\overline{X15} \bullet M15 + M15 \bullet R15 + R15 + \overline{X15}$   
Set if the absolute value of the content of memory is larger than the absolute value of the index register; cleared otherwise.

### Addressing Modes, Machine Code, and Execution Times:

Source Form	Address Mode	Object Code	Cycles	Access Detail
CPX #opr16i	IMM	8E jj kk	2	OP
CPX opr8a	DIR	9E dd	3	RfP
CPX opr16a	EXT	BE hh ll	3	ROP
CPX oprx0,xysp	IDX	AE xb	3	RfP
CPX oprx9,xysp	IDX1	AE xb ff	3	RPO
CPX oprx16,xysp	IDX2	AE xb ee ff	4	fRPP
CPX [D,xysp]	[D,IDX]	AE xb	6	fIfRfP
CPX [oprx16,xysp]	[IDX2]	AE xb ee ff	6	fIPRfP

# CPY

## Compare Index Register Y

# CPY

**Operation:**  $(Y) - (M : M + 1)$

**Description:** Compares the content of index register Y to a 16-bit value at the address specified, and sets the condition codes accordingly. The compare is accomplished internally by a 16-bit subtract of  $(M : M + 1)$  from Y without modifying either Y or  $(M : M + 1)$ .

**Condition Codes and Boolean Formulas:**

S	X	H	I	N	Z	V	C
-	-	-	-	$\Delta$	$\Delta$	$\Delta$	$\Delta$

- N: Set if MSB of result is set; cleared otherwise.
- Z: Set if result is \$0000; cleared otherwise.
- V:  $Y15 \bullet \overline{M15} \bullet \overline{R15} + \overline{Y15} \bullet M15 \bullet R15$   
Set if two's complement overflow resulted from the operation; cleared otherwise.
- C:  $\overline{Y15} \bullet M15 + M15 \bullet R15 + R15 + \overline{Y15}$   
Set if the absolute value of the content of memory is larger than the absolute value of the index register; cleared otherwise.

**Addressing Modes, Machine Code, and Execution Times:**

Source Form	Address Mode	Object Code	Cycles	Access Detail
CPY #opr16i	IMM	8D jj kk	2	OP
CPY opr8a	DIR	9D dd	3	RfP
CPY opr16a	EXT	BD hh ll	3	ROP
CPY oprx0_xysp	IDX	AD xb	3	RfP
CPY oprx9_xysp	IDX1	AD xb ff	3	RPO
CPY oprx16_xysp	IDX2	AD xb ee ff	4	fRPP
CPY [D,xysp]	[D,IDX]	AD xb	6	fIfRfP
CPY [opr16,xysp]	[IDX2]	AD xb ee ff	6	fIPRfP

# DAA

## Decimal Adjust A

# DAA

**Description:** DAA adjusts the content of accumulator A and the state of the C status bit to represent the correct binary-coded-decimal sum and the associated carry when a BCD calculation has been performed. In order to execute DAA, the content of accumulator A, the state of the C status bit, and the state of the H status bit must all be the result of performing an ABA, ADD or ADC on BCD operands, with or without an initial carry.

The table below shows DAA operation for all legal combinations of input operands. Columns 1 through 4 represent the results of ABA, ADC, or ADD operations on BCD operands. The correction factor in column 5 is added to the accumulator to restore the result of an operation on two BCD operands to a valid BCD value, and to set or clear the C bit. All values are in hexadecimal.

1	2	3	4	5	6
Initial C Bit Value	Value of A[7:4]	Initial H Bit Value	Value of A[3:0]	Correction Factor	Corrected C Bit Value
0	0-9	0	0-9	00	0
0	0-8	0	A-F	06	0
0	0-9	1	0-3	06	0
0	A-F	0	0-9	60	1
0	9-F	0	A-F	66	1
0	A-F	1	0-3	66	1
1	0-2	0	0-9	60	1
1	0-2	0	A-F	66	1
1	0-3	1	0-3	66	1

### Condition Codes and Boolean Formulas:

S	X	H	I	N	Z	V	C
-	-	-	-	Δ	Δ	?	Δ

- N: Set if MSB of result is set; cleared otherwise.  
Z: Set if result is \$00; cleared otherwise.  
V: Undefined.  
C: Represents BCD carry. See table above.

### Addressing Modes, Machine Code, and Execution Times:

Source Form	Address Mode	Object Code	Cycles	Access Detail
DAA	INH	18 07	3	OEO

# DBEQ Decrement and Branch if Equal to Zero DBEQ

**Operation:** (Counter) – 1  $\Rightarrow$  Counter  
If (Counter) = 0, then (PC) + \$0003 + Rel  $\Rightarrow$  PC,

**Description:** Subtract one from the specified counter register A, B, D, X, Y, or SP. If the counter register has reached zero, execute a branch to the specified relative destination. The DBEQ instruction is encoded into three bytes of machine code including the 9-bit relative offset (–256 to +255 locations from the start of the next instruction).

IBEQ and TBEQ instructions are similar to DBEQ except that the counter is incremented or tested rather than being decremented. Bits 7 and 6 of the instruction postbyte are used to determine which operation is to be performed.

## Condition Codes and Boolean Formulas:

S	X	H	I	N	Z	V	C
–	–	–	–	–	–	–	–

None affected.

## Addressing Modes, Machine Code, and Execution Times:

Source Form	Address Mode	Object Code <sup>1</sup>	Cycles	Access Detail
DBEQ <i>abdxys</i> , <i>rel9</i>	REL	04 1b rr	3/3	PPP

Notes:

1. Encoding for 1b is summarized in the following table. Bit 3 is not used (don't care), bit 5 selects branch on zero (DBEQ – 0) or not zero (DBNE – 1) versions, and bit 4 is the sign bit of the 9-bit relative offset. Bits 7 and 6 would be 0:0 for DBEQ.

Count Register	Bits 2:0	Source Form	Object Code (if offset is positive)	Object Code (if offset is negative)
A	000	DBEQ A, <i>rel9</i>	04 00 rr	04 10 rr
B	001	DBEQ B, <i>rel9</i>	04 01 rr	04 11 rr
D	100	DBEQ D, <i>rel9</i>	04 04 rr	04 14 rr
X	101	DBEQ X, <i>rel9</i>	04 05 rr	04 15 rr
Y	110	DBEQ Y, <i>rel9</i>	04 06 rr	04 16 rr
SP	111	DBEQ SP, <i>rel9</i>	04 07 rr	04 17 rr

# DBNE Decrement and Branch if Not Equal to Zero DBNE

**Operation:** (Counter) – 1  $\Rightarrow$  Counter  
If (Counter) not = 0, then (PC) + \$0003 + Rel  $\Rightarrow$  PC,

**Description:** Subtract one from the specified counter register A, B, D, X, Y, or SP. If the counter register has not been decremented to zero, execute a branch to the specified relative destination. The DBNE instruction is encoded into three bytes of machine code including a 9-bit relative offset (–256 to +255 locations from the start of the next instruction).

IBNE and TBNE instructions are similar to DBNE except that the counter is incremented or tested rather than being decremented. Bits 7 and 6 of the instruction postbyte are used to determine which operation is to be performed.

## Condition Codes and Boolean Formulas:

S	X	H	I	N	Z	V	C
–	–	–	–	–	–	–	–

None affected.

## Addressing Modes, Machine Code, and Execution Times:

Source Form	Address Mode	Object Code <sup>1</sup>	Cycles	Access Detail
DBNE <i>abdxys, rel9</i>	REL	04 1b rr	3/3	PPP

Notes:

1. Encoding for 1b is summarized in the following table. Bit 3 is not used (don't care), bit 5 selects branch on zero (DBEQ – 0) or not zero (DBNE – 1) versions, and bit 4 is the sign bit of the 9-bit relative offset. Bits 7 and 6 would be 0:0 for DBNE.

Count Register	Bits 2:0	Source Form	Object Code (if offset is positive)	Object Code (if offset is negative)
A	000	DBNE A, <i>rel9</i>	04 20 rr	04 30 rr
B	001	DBNE B, <i>rel9</i>	04 21 rr	04 31 rr
D	100	DBNE D, <i>rel9</i>	04 24 rr	04 34 rr
X	101	DBNE X, <i>rel9</i>	04 25 rr	04 35 rr
Y	110	DBNE Y, <i>rel9</i>	04 26 rr	04 36 rr
SP	111	DBNE SP, <i>rel9</i>	04 27 rr	04 37 rr

# DEC

## Decrement Memory

# DEC

**Operation:** (M) – \$01 ⇒ M

**Description:** Subtract one from the content of memory location M.

The N, Z and V status bits are set or cleared according to the results of the operation. The C status bit is not affected by the operation, thus allowing the DEC instruction to be used as a loop counter in multiple-precision computations.

### Condition Codes and Boolean Formulas:

S	X	H	I	N	Z	V	C
–	–	–	–	Δ	Δ	Δ	–

N: Set if MSB of result is set; cleared otherwise.

Z: Set if result is \$00; cleared otherwise.

V: Set if there was a two's complement overflow as a result of the operation; cleared otherwise. Two's complement overflow occurs if and only if (M) was \$80 before the operation.

### Addressing Modes, Machine Code, and Execution Times:

Source Form	Address Mode	Object Code	Cycles	Access Detail
DEC <i>opr16a</i>	EXT	73 hh ll	4	rOPw
DEC <i>opr0_xysp</i>	IDX	63 xb	3	rPw
DEC <i>opr9,xysp</i>	IDX1	63 xb ff	4	rPOw
DEC <i>opr16,xysp</i>	IDX2	63 xb ee ff	5	frPPw
DEC [D, <i>xysp</i> ]	[D,IDX]	63 xb	6	fifrPw
DEC [ <i>opr16,xysp</i> ]	[IDX2]	63 xb ee ff	6	fIPrPw

# DECA

Decrement A

# DECA

**Operation:**  $(A) - \$01 \Rightarrow A$

**Description:** Subtract one from the content of accumulator A.

The N, Z and V status bits are set or cleared according to the results of the operation. The C status bit is not affected by the operation, thus allowing the DEC instruction to be used as a loop counter in multiple-precision computations.

## Condition Codes and Boolean Formulas:

S	X	H	I	N	Z	V	C
-	-	-	-	$\Delta$	$\Delta$	$\Delta$	-

N: Set if MSB of result is set; cleared otherwise.

Z: Set if result is \$00; cleared otherwise.

V: Set if there was a two's complement overflow as a result of the operation; cleared otherwise. Two's complement overflow occurs if and only if (A) was \$80 before the operation.

## Addressing Modes, Machine Code, and Execution Times:

Source Form	Address Mode	Object Code	Cycles	Access Detail
DECA	INH	43	1	0



# DECB

Decrement B

# DECB

**Operation:**  $(B) - \$01 \Rightarrow B$

**Description:** Subtract one from the content of accumulator B.

The N, Z and V status bits are set or cleared according to the results of the operation. The C status bit is not affected by the operation, thus allowing the DEC instruction to be used as a loop counter in multiple-precision computations.

## Condition Codes and Boolean Formulas:

S	X	H	I	N	Z	V	C
-	-	-	-	$\Delta$	$\Delta$	$\Delta$	-

N: Set if MSB of result is set; cleared otherwise.

Z: Set if result is \$00; cleared otherwise.

V: Set if there was a two's complement overflow as a result of the operation; cleared otherwise. Two's complement overflow occurs if and only if (B) was \$80 before the operation.

## Addressing Modes, Machine Code, and Execution Times:

Source Form	Address Mode	Object Code	Cycles	Access Detail
DECB	INH	53	1	0

# DES

## Decrement Stack Pointer

# DES

**Operation:** (SP) – \$0001 ⇒ SP

**Description:** Subtract one from the SP. This instruction assembles to LEAS –1,SP. The LEAS instruction does not affect condition codes as DEX or DEY instructions do.

**Condition Codes and Boolean Formulas:**

S	X	H	I	N	Z	V	C
–	–	–	–	–	–	–	–

None affected.

**Addressing Modes, Machine Code, and Execution Times:**

Source Form	Address Mode	Object Code	Cycles	Access Detail
DES translates to... LEAS –1,SP	IDX	1B 9F	2	PP1

**Notes:**

1. Due to internal CPU requirements, the program word fetch is performed twice to the same address during this instruction.

# DEX

Decrement Index Register X

# DEX

**Operation:**  $(X) - \$0001 \Rightarrow X$

**Description:** Subtract one from index register X. LEAX -1,X can produce the same result, but LEAX does not affect the Z bit. Although the LEAX instruction is more flexible, DEX requires only one byte of object code.

Only the Z bit is set or cleared according to the result of this operation.

**Condition Codes and Boolean Formulas:**

S	X	H	I	N	Z	V	C
-	-	-	-	-	$\Delta$	-	-

Z: Set if result is \$0000; cleared otherwise.

**Addressing Modes, Machine Code, and Execution Times:**

Source Form	Address Mode	Object Code	Cycles	Access Detail
DEX	INH	09	1	0

# DEY

## Decrement Index Register Y

# DEY

**Operation:**  $(Y) - \$0001 \Rightarrow Y$

**Description:** Subtract one from index register Y. LEAY -1,Y can produce the same result, but LEAY does not affect the Z bit. Although the LEAY instruction is more flexible, DEY requires only one byte of object code.

Only the Z bit is set or cleared according to the result of this operation.

**Condition Codes and Boolean Formulas:**

S	X	H	I	N	Z	V	C
-	-	-	-	-	$\Delta$	-	-

Z: Set if result is \$0000; cleared otherwise.

**Addressing Modes, Machine Code, and Execution Times:**

Source Form	Address Mode	Object Code	Cycles	Access Detail
DEY	INH	03	1	0

# EDIV

## Extended Divide 32-Bit by 16-Bit (Unsigned)

# EDIV

**Operation:**  $(Y : D) \div (X) \Rightarrow Y; \text{Remainder} \Rightarrow D$

**Description:** Divides a 32-bit unsigned dividend by a 16-bit divisor, producing a 16-bit unsigned quotient and an unsigned 16-bit remainder. All operands and results are located in CPU registers. If an attempt to divide by zero is made, the contents of double accumulator D and index register Y do not change, but the states of the N and Z bits in the CCR are undefined.

### Condition Codes and Boolean Formulas:

S	X	H	I	N	Z	V	C
-	-	-	-	$\Delta$	$\Delta$	$\Delta$	$\Delta$

N: Set if MSB of result is set; cleared otherwise. Undefined after overflow or division by zero.

Z: Set if result is \$0000; cleared otherwise. Undefined after overflow or division by zero.

V: Set if the result was > \$FFFF; cleared otherwise. Undefined after division by zero.

C: Set if divisor was \$0000; cleared otherwise.

### Addressing Modes, Machine Code, and Execution Times:

Source Form	Address Mode	Object Code	Cycles	Access Detail
EDIV	INH	11	11	fffffffffe0

# EDIVS

## Extended Divide 32-Bit by 16-Bit (Signed)

# EDIVS

**Operation:**  $(Y : D) \div (X) \Rightarrow Y; \text{Remainder} \Rightarrow D$

**Description:** Divides a signed 32-bit dividend by a 16-bit signed divisor, producing a signed 16-bit quotient and a signed 16-bit remainder. All operands and results are located in CPU registers. If an attempt to divide by zero is made, the C status bit is set and the contents of double accumulator D and index register Y do not change, but the states of the N and Z bits in the CCR are undefined.

### Condition Codes and Boolean Formulas:

S	X	H	I	N	Z	V	C
—	—	—	—	$\Delta$	$\Delta$	$\Delta$	$\Delta$

N: Set if MSB of result is set; cleared otherwise. Undefined after overflow or division by zero.

Z: Set if result is \$0000; cleared otherwise. Undefined after overflow or division by zero.

V: Set if the result was > \$7FFF or < \$8000; cleared otherwise. Undefined after division by zero.

C: Set if divisor was \$0000; cleared otherwise. (Indicates division by zero.)

### Addressing Modes, Machine Code, and Execution Times:

Source Form	Address Mode	Object Code	Cycles	Access Detail
EDIVS	INH	18 14	12	0fffffffe0

EMACS

Extended Multiply and Accumulate  
(Signed)  
16-Bit by 16-Bit to 32-Bit

EMACS

**Operation:**  $(M(X) : M(X+1)) \times (M(Y) : M(Y+1)) + (M \sim M+3) \Rightarrow M \sim M+3$

**Description:** A 16-bit value is multiplied by a 16-bit value to produce a 32-bit intermediate result. This 32-bit intermediate result is then added to the content of a 32-bit accumulator in memory. EMACS is a signed integer operation. All operands and results are located in memory. When the EMACS instruction is executed, the first source operand is fetched from an address pointed to by X, and the second source operand is fetched from an address pointed to by index register Y. Before the instruction is executed, the X and Y index registers must contain values that point to the most significant bytes of the source operands. The most significant byte of the 32-bit result is specified by an extended address supplied with the instruction.

**Condition Codes and Boolean Formulas:**

S	X	H	I	N	Z	V	C
-	-	-	-	$\Delta$	$\Delta$	$\Delta$	$\Delta$

- N: Set if MSB of result is set; cleared otherwise.
- Z: Set if result is \$00000000; cleared otherwise.
- V:  $M31 \bullet I31 \bullet \overline{R31} + \overline{M31} \bullet I31 \bullet R31$   
Set if result > \$7FFFFFFF (+ overflow) or < \$80000000 (- underflow).  
Indicates two's complement overflow.
- C:  $M15 \bullet I15 + I15 \bullet \overline{R15} + \overline{R15} \bullet M15$   
Set if there was a carry from bit 15 of the result; cleared otherwise.  
Indicates a carry from low word to high word of the result occurred.

**Addressing Modes, Machine Code, and Execution Times:**

Source Form <sup>1</sup>	Address Mode	Object Code	Cycles	Access Detail
EMACS <i>opr16a</i>	Special	18 12 hh 11	13	ORROffERRÉWWP

Notes:  
1. *opr16a* is an extended address specification. Both X and Y point to source operands.

# EMAXD

Place Larger of Two  
Unsigned 16-Bit Values  
in Accumulator D

# EMAXD

**Operation:**  $\text{MAX} ((D), (M : M + 1)) \Rightarrow D$

**Description:** Subtracts an unsigned 16-bit value in memory from an unsigned 16-bit value in double accumulator D to determine which is larger, and leaves the larger of the two values in D. The Z status bit is set when the result of the subtraction is zero (the values are equal), and the C status bit is set when the subtraction requires a borrow (the value in memory is larger than the value in the accumulator). When C = 1, the value in D has been replaced by the value in memory.

The unsigned value in memory is accessed by means of indexed addressing modes, which allow a great deal of flexibility in specifying the address of the operand. Auto increment/decrement variations of indexed addressing facilitate finding the largest value in a list of values.

## Condition Codes and Boolean Formulas:

S	X	H	I	N	Z	V	C
—	—	—	—	Δ	Δ	Δ	Δ

N: Set if MSB of result is set; cleared otherwise.

Z: Set if result is \$0000; cleared otherwise.

V:  $D15 \bullet \overline{M15} \bullet \overline{R15} + \overline{D15} \bullet M15 \bullet R15$   
Set if a two's complement overflow resulted from the operation; cleared otherwise.

C:  $\overline{D15} \bullet M15 + M15 \bullet R15 + R15 \bullet \overline{D15}$   
Set if the value of the content of memory is larger than the value of the accumulator; cleared otherwise.

Condition codes reflect internal subtraction ( $R = D - M : M + 1$ ).

## Addressing Modes, Machine Code, and Execution Times:

Source Form	Address Mode	Object Code	Cycles	Access Detail
EMAXD <i>opr0_xysp</i>	IDX	18 1A xb	4	ORfP
EMAXD <i>opr9_xysp</i>	IDX1	18 1A xb ff	4	ORPO
EMAXD <i>opr16_xysp</i>	IDX2	18 1A xb ee ff	5	OfRPP
EMAXD [ <i>D,xysp</i> ]	[D,IDX]	18 1A xb	7	OfIfRfP
EMAXD [ <i>opr16,xysp</i> ]	[IDX2]	18 1A xb ee ff	7	OfIPRfP



# EMAXM

Place Larger of Two  
Unsigned 16-Bit Values  
in Memory

# EMAXM

**Operation:**  $\text{MAX} ((D), (M : M + 1)) \Rightarrow M : M + 1$

**Description:** Subtracts an unsigned 16-bit value in memory from an unsigned 16-bit value in double accumulator D to determine which is larger, and leaves the larger of the two values in the memory location. The Z status bit is set when the result of the subtraction is zero (the values are equal), and the C status bit is set when the subtraction requires a borrow (the value in memory is larger than the value in the accumulator). When C = 0, the value in D has replaced the value in memory.

The unsigned value in memory is accessed by means of indexed addressing modes, which allow a great deal of flexibility in specifying the address of the operand.

## Condition Codes and Boolean Formulas:

S	X	H	I	N	Z	V	C
-	-	-	-	$\Delta$	$\Delta$	$\Delta$	$\Delta$

N: Set if MSB of result is set; cleared otherwise.

Z: Set if result is \$0000; cleared otherwise.

V:  $D15 \bullet \bar{M}15 \bullet R15 + \bar{D}15 \bullet M15 \bullet R15$   
Set if a two's complement overflow resulted from the operation; cleared otherwise.

C:  $\bar{D}15 \bullet M15 + M15 \bullet R15 + R15 \bullet \bar{D}15$   
Set if the value of the content of memory is larger than the value of the accumulator; cleared otherwise.

Condition codes reflect internal subtraction ( $R = D - M : M + 1$ ).

## Addressing Modes, Machine Code, and Execution Times:

Source Form	Address Mode	Object Code	Cycles	Access Detail
EMAXM <i>opr0_xysp</i>	IDX	18 1E xb	4	ORPW
EMAXM <i>opr9,xysp</i>	IDX1	18 1E xb ff	5	ORPWO
EMAXM <i>opr16,xysp</i>	IDX2	18 1E xb ee ff	6	OfRPWP
EMAXM [D,xysp]	[D,IDX]	18 1E xb	7	OfIRFPW
EMAXM [ <i>opr16,xysp</i> ]	[IDX2]	18 1E xb ee ff	7	OfIPRPW

# EMIND

Place Smaller of Two  
Unsigned 16-Bit Values  
in Accumulator D

# EMIND

**Operation:**  $\text{MIN}((D), (M : M + 1)) \Rightarrow D$

**Description:** Subtracts an unsigned 16-bit value in memory from an unsigned 16-bit value in double accumulator D to determine which is larger, and leaves the smaller of the two values in D. The Z status bit is set when the result of the subtraction is zero (the values are equal), and the C status bit is set when the subtraction requires a borrow (the value in memory is larger than the value in the accumulator). When C = 0, the value in D has been replaced by the value in memory.

The unsigned value in memory is accessed by means of indexed addressing modes, which allow a great deal of flexibility in specifying the address of the operand. Auto increment/decrement variations of indexed addressing facilitate finding the largest value in a list of values.

## Condition Codes and Boolean Formulas:

S	X	H	I	N	Z	V	C
—	—	—	—	Δ	Δ	Δ	Δ

N: Set if MSB of result is set; cleared otherwise.

Z: Set if result is \$0000; cleared otherwise.

V:  $D15 \bullet \overline{M15} \bullet \overline{R15} + \overline{D15} \bullet M15 \bullet R15$   
Set if a two's complement overflow resulted from the operation;  
cleared otherwise.

C:  $\overline{D15} \bullet M15 + M15 \bullet R15 + R15 \bullet \overline{D15}$   
Set if the value of the content of memory is larger than the value of  
the accumulator; cleared otherwise.

Condition codes reflect internal subtraction ( $R = D - M : M + 1$ ).

## Addressing Modes, Machine Code, and Execution Times:

Source Form	Address Mode	Object Code	Cycles	Access Detail
EMIND <i>opr0,xysp</i>	IDX	18 1B xb	4	ORfP
EMIND <i>opr9,xysp</i>	IDX1	18 1B xb ff	4	ORPO
EMIND <i>opr16,xysp</i>	IDX2	18 1B xb ee ff	5	OfRPP
EMIND [ <i>D,xysp</i> ]	[D,IDX]	18 1B xb	7	OfIfRfP
EMIND [ <i>opr16,xysp</i> ]	[IDX2]	18 1B xb ee ff	7	OfIPRfP

# EMINM

Place Smaller of Two  
Unsigned 16-Bit Values  
in Memory

# EMINM

**Operation:**  $\text{MIN} ((D), (M : M + 1)) \Rightarrow M : M + 1$

**Description:** Subtracts an unsigned 16-bit value in memory from an unsigned 16-bit value in double accumulator D to determine which is larger, and leaves the smaller of the two values in the memory location. The Z status bit is set when the result of the subtraction is zero (the values are equal), and the C status bit is set when the subtraction requires a borrow (the value in memory is larger than the value in the accumulator). When C = 1, the value in D has replaced the value in memory.

The unsigned value in memory is accessed by means of indexed addressing modes, which allow a great deal of flexibility in specifying the address of the operand.

## Condition Codes and Boolean Formulas:

S	X	H	I	N	Z	V	C
-	-	-	-	$\Delta$	$\Delta$	$\Delta$	$\Delta$

N: Set if MSB of result is set; cleared otherwise.

Z: Set if result is \$0000; cleared otherwise.

V:  $D15 \bullet M15 \bullet R15 + \overline{D15} \bullet M15 \bullet R15$   
Set if a two's complement overflow resulted from the operation; cleared otherwise.

C:  $\overline{D15} \bullet M15 + M15 \bullet R15 + R15 \bullet \overline{D15}$   
Set if the value of the content of memory is larger than the value of the accumulator; cleared otherwise.

Condition codes reflect internal subtraction ( $R = D - M : M + 1$ ).

## Addressing Modes, Machine Code, and Execution Times:

Source Form	Address Mode	Object Code	Cycles	Access Detail
EMINM <i>opr</i> x0_ <i>xy</i> sp	IDX	18 1F xb	4	ORPW
EMINM <i>opr</i> x9, <i>xy</i> sp	IDX1	18 1F xb ff	5	ORPWO
EMINM <i>opr</i> x16, <i>xy</i> sp	IDX2	18 1F xb ee ff	6	OFRFPW
EMINM [D, <i>xy</i> sp]	[D,IDX]	18 1F xb	7	OIFRFPW
EMINM [ <i>opr</i> x16, <i>xy</i> sp]	[IDX2]	18 1F xb ee ff	7	OIFRFPW

# EMUL

## Extended Multiply 16-Bit by 16-Bit (Unsigned)

# EMUL

**Operation:**  $(D) \times (Y) \Rightarrow Y : D$

**Description:** An unsigned 16-bit value is multiplied by an unsigned 16-bit value to produce an unsigned 32-bit result. The first source operand must be loaded into 16-bit double accumulator D and the second source operand must be loaded into index register Y before executing the instruction. When the instruction is executed, the value in D is multiplied by the value in Y. The upper 16-bits of the 32-bit result are stored in Y and the low-order 16-bits of the result are stored in D.

The C status bit can be used to round the high-order 16 bits of the result.

### Condition Codes and Boolean Formulas:

S	X	H	I	N	Z	V	C
—	—	—	—	$\Delta$	$\Delta$	—	$\Delta$

N: Set if the MSB of the result is set; cleared otherwise.

Z: Set if result is \$00000000; cleared otherwise.

C: Set if bit 15 of the result is set; cleared otherwise.

### Addressing Modes, Machine Code, and Execution Times:

Source Form	Address Mode	Object Code	Cycles	Access Detail
EMUL	INH	13	3	ff0

# EMULS

Extended Multiply  
16-Bit by 16-Bit (Signed)

# EMULS

**Operation:**  $(D) \times (Y) \Rightarrow Y : D$

**Description:** A signed 16-bit value is multiplied by a signed 16-bit value to produce a signed 32-bit result. The first source operand must be loaded into 16-bit double accumulator D and the second source operand must be loaded into index register Y before executing the instruction. When the instruction is executed, D is multiplied by the value Y. The 16 high-order bits of the 32-bit result are stored in Y and the 16 low-order bits of the result are stored in D.

The C status bit can be used to round the high-order 16 bits of the result.

**Condition Codes and Boolean Formulas:**

S	X	H	I	N	Z	V	C
-	-	-	-	$\Delta$	$\Delta$	-	$\Delta$

N: Set if the MSB of the result is set; cleared otherwise.

Z: Set if result is \$00000000; cleared otherwise.

C: Set if bit 15 of the result is set; cleared otherwise.

**Addressing Modes, Machine Code, and Execution Times:**

Source Form	Address Mode	Object Code	Cycles	Access Detail
EMULS	INH	18 13	3	OFO

# EORA

Exclusive-OR A

# EORA

**Operation:**  $(A) \oplus (M) \Rightarrow A$

**Description:** Performs the logical exclusive OR between the content of accumulator A and the content of memory location M. The result is placed in A. Each bit of A after the operation is the logical exclusive OR of the corresponding bits of M and A before the operation.

**Condition Codes and Boolean Formulas:**

S	X	H	I	N	Z	V	C
-	-	-	-	$\Delta$	$\Delta$	0	-

N: Set if MSB of result is set; cleared otherwise.

Z: Set if result is \$00; cleared otherwise.

V: 0; Cleared.

**Addressing Modes, Machine Code, and Execution Times:**

Source Form	Address Mode	Object Code	Cycles	Access Detail
EORA #opr8i	IMM	88 ii	1	P
EORA opr8a	DIR	98 dd	3	rFP
EORA opr16a	EXT	B8 hh ll	3	rOP
EORA oprx0,xysp	IDX	A8 xb	3	rFP
EORA oprx9,xysp	IDX1	A8 xb ff	3	rPO
EORA oprx16,xysp	IDX2	A8 xb ee ff	4	frPP
EORA [D,xysp]	[D,IDX]	A8 xb	6	fIfrfP
EORA [oprx16,xysp]	[IDX2]	A8 xb ee ff	6	fIPrfP

# EORB

Exclusive-OR B

# EORB

**Operation:**  $(B) \oplus (M) \Rightarrow B$

**Description:** Performs the logical exclusive OR between the content of accumulator B and the content of memory location M. The result is placed in A. Each bit of A after the operation is the logical exclusive OR of the corresponding bits of M and B before the operation.

**Condition Codes and Boolean Formulas:**

S	X	H	I	N	Z	V	C
-	-	-	-	$\Delta$	$\Delta$	0	-

- N: Set if MSB of result is set; cleared otherwise.  
Z: Set if result is \$00; cleared otherwise.  
V: 0; Cleared.

**Addressing Modes, Machine Code, and Execution Times:**

Source Form	Address Mode	Object Code	Cycles	Access Detail
EORB #opr8i	IMM	C8 ii	1	P
EORB opr8a	DIR	D8 dd	3	rFP
EORB opr16a	EXT	F8 hh ll	3	rOP
EORB oprx0_xysp	IDX	E8 xb	3	rFP
EORB oprx9_xysp	IDX1	E8 xb ff	3	rPO
EORB oprx16_xysp	IDX2	E8 xb ee ff	4	frPP
EORB [D,xysp]	[D,IDX]	E8 xb	6	fIfrfP
EORB [opr16,xysp]	[IDX2]	E8 xb ee ff	6	fIPrfP

# ETBL

## Extended Table Lookup and Interpolate

# ETBL

**Operation:**  $(M : M + 1) + [(B) \times ((M + 2 : M + 3) - (M : M + 1))] \Rightarrow D$

**Description:** ETBL linearly interpolates one of 256 result values that fall between each pair of data entries in a lookup table stored in memory. Data points in the table represent the endpoints of equally-spaced line segments. Table entries and the interpolated result are 16-bit values. The result is stored in the D accumulator.

Before executing ETBL, set up an index register so that it points to the starting point (X1) of a line segment when the instruction is executed. X1 is the table entry closest to, but less than or equal to, the desired lookup value. The next table entry after X1 is X2. XL is the distance in X between X1 and X2. Load accumulator B with a binary fraction (radix point to left of MSB) representing the ratio  $(XL - X1) \div (X2 - X1)$ .

The 16-bit unrounded result is calculated using the following expression:

$$D = Y1 + [(B) \times (Y2 - Y1)]$$

Where

$$(B) = (XL - X1) \div (X2 - X1)$$

Y1 = 16-bit data entry pointed to by <effective address>

Y2 = 16-bit data entry pointed to by <effective address> + 2

The intermediate value  $[(B) \times (Y2 - Y1)]$  produces a 24-bit result with the radix point between bits 7 and 8. Any indexed addressing mode, except indirect modes or 9-bit and 16-bit offset modes, can be used to identify the first data point (X1, Y1). The second data point is the next table entry.

### Condition Codes and Boolean Formulas:

S	X	H	I	N	Z	V	C
-	-	-	-	Δ	Δ	-	?

N: Set if MSB of result is set; cleared otherwise.

Z: Set if result is \$0000; cleared otherwise.

C: Undefined.

### Addressing Modes, Machine Code, and Execution Times:

Source Form	Address Mode	Object Code	Cycles	Access Detail
ETBL <i>opr0_xysp</i>	IDX	18 3F xb	10	ORRfffffp



# EXG

## Exchange Register Contents

# EXG

**Operation:** See table

**Description:** Exchanges the contents of registers specified in the instruction as shown below. Note that the order in which exchanges between 8-bit and 16-bit registers are specified affects the high byte of the 16-bit registers differently. Exchanges of D with A or B are ambiguous. Cases involving TMP2 and TMP3 are reserved for Motorola use, so some assemblers may not permit their use, but it is possible to generate these cases by using DC.B or DC.W assembler directives.

### Condition Codes and Boolean Formulas:

S	X	H	I	N	Z	V	C
—	—	—	—	—	—	—	—

None affected, unless the CCR is the destination register. Condition codes take on the value of the corresponding source bits, except that the X mask bit cannot change from zero to one. Software can leave the X bit set, leave it cleared, or change it from one to zero, but it can only be set by a reset or by recognition of an  $\overline{\text{XIRQ}}$  interrupt.

### Addressing Modes, Machine Code, and Execution Times:

Source Form	Address Mode	Object Code <sup>1</sup>	Cycles	Access Detail
EXG <i>abcdxys,abcdxys</i>	INH	B7 eb	1	P

Notes:

1. Legal coding for eb is summarized in the following table. Columns represent the high-order source digit. Rows represent the low-order destination digit (bit 3 is a don't-care). Values are in hexadecimal.

	8	9	A	B	C	D	E	F
0	A $\leftrightarrow$ A	B $\leftrightarrow$ A	CCR $\leftrightarrow$ A	TMP3 <sub>L</sub> $\Rightarrow$ A \$00:A $\Rightarrow$ TMP3	B $\Rightarrow$ A A $\Rightarrow$ B	X <sub>L</sub> $\Rightarrow$ A \$00:A $\Rightarrow$ X	Y <sub>L</sub> $\Rightarrow$ A \$00:A $\Rightarrow$ Y	SP <sub>L</sub> $\Rightarrow$ A \$00:A $\Rightarrow$ SP
1	A $\leftrightarrow$ B	B $\leftrightarrow$ B	CCR $\leftrightarrow$ B	TMP3 <sub>L</sub> $\Rightarrow$ B \$FF:B $\Rightarrow$ TMP3	B $\Rightarrow$ B \$FF $\Rightarrow$ A	X <sub>L</sub> $\Rightarrow$ B \$FF:B $\Rightarrow$ X	Y <sub>L</sub> $\Rightarrow$ B \$FF:B $\Rightarrow$ Y	SP <sub>L</sub> $\Rightarrow$ B \$FF:B $\Rightarrow$ SP
2	A $\leftrightarrow$ CCR	B $\leftrightarrow$ CCR	CCR $\leftrightarrow$ CCR	TMP3 <sub>L</sub> $\Rightarrow$ CCR \$FF:CCR $\Rightarrow$ TMP3	B $\Rightarrow$ CCR \$FF:CCR $\Rightarrow$ D	X <sub>L</sub> $\Rightarrow$ CCR \$FF:CCR $\Rightarrow$ X	Y <sub>L</sub> $\Rightarrow$ CCR \$FF:CCR $\Rightarrow$ Y	SP <sub>L</sub> $\Rightarrow$ CCR \$FF:CCR $\Rightarrow$ SP
3	\$00:A $\Rightarrow$ TMP2 TMP2 <sub>L</sub> $\Rightarrow$ A	\$00:B $\Rightarrow$ TMP2 TMP2 <sub>L</sub> $\Rightarrow$ B	\$00:CCR $\Rightarrow$ TMP2 TMP2 <sub>L</sub> $\Rightarrow$ CCR	TMP3 $\leftrightarrow$ TMP2	D $\leftrightarrow$ TMP2	X $\leftrightarrow$ TMP2	Y $\leftrightarrow$ TMP2	SP $\leftrightarrow$ TMP2
4	\$00:A $\Rightarrow$ D	\$00:B $\Rightarrow$ D	\$00:CCR $\Rightarrow$ D B $\Rightarrow$ CCR	TMP3 $\leftrightarrow$ D	D $\leftrightarrow$ D	X $\leftrightarrow$ D	Y $\leftrightarrow$ D	SP $\leftrightarrow$ D
5	\$00:A $\Rightarrow$ X X <sub>L</sub> $\Rightarrow$ A	\$00:B $\Rightarrow$ X X <sub>L</sub> $\Rightarrow$ B	\$00:CCR $\Rightarrow$ X X <sub>L</sub> $\Rightarrow$ CCR	TMP3 $\leftrightarrow$ X	D $\leftrightarrow$ X	X $\leftrightarrow$ X	Y $\leftrightarrow$ X	SP $\leftrightarrow$ X
6	\$00:A $\Rightarrow$ Y Y <sub>L</sub> $\Rightarrow$ A	\$00:B $\Rightarrow$ Y Y <sub>L</sub> $\Rightarrow$ B	\$00:CCR $\Rightarrow$ Y Y <sub>L</sub> $\Rightarrow$ CCR	TMP3 $\leftrightarrow$ Y	D $\leftrightarrow$ Y	X $\leftrightarrow$ Y	Y $\leftrightarrow$ Y	SP $\leftrightarrow$ Y
7	\$00:A $\Rightarrow$ SP SP <sub>L</sub> $\Rightarrow$ A	\$00:B $\Rightarrow$ SP SP <sub>L</sub> $\Rightarrow$ B	\$00:CCR $\Rightarrow$ SP SP <sub>L</sub> $\Rightarrow$ CCR	TMP3 $\leftrightarrow$ SP	D $\leftrightarrow$ SP	X $\leftrightarrow$ SP	Y $\leftrightarrow$ SP	SP $\leftrightarrow$ SP

# FDIV

## Fractional Divide

# FDIV

**Operation:** (D) ÷ (X) ⇒ X; Remainder ⇒ D

**Description:** Divides an unsigned 16-bit numerator in double accumulator D by an unsigned 16-bit denominator in index register X, producing an unsigned 16-bit quotient in X, and an unsigned 16-bit remainder in D. If both the numerator and the denominator are assumed to have radix points in the same positions, the radix point of the quotient is to the left of bit 15. The numerator must be less than the denominator. In the case of overflow (denominator is less than or equal to the numerator) or division by zero, the quotient is set to \$FFFF, and the remainder is indeterminate.

FDIV is equivalent to multiplying the numerator by 2<sup>16</sup> and then performing 32 x 16-bit integer division. The result is interpreted as a binary-weighted fraction, which resulted from the division of a 16-bit integer by a larger 16-bit integer. A result of \$0001 corresponds to 0.000015, and \$FFFF corresponds to 0.9998. The remainder of an IDIV instruction can be resolved into a binary-weighted fraction by an FDIV instruction. The remainder of an FDIV instruction can be resolved into the next 16 bits of binary-weighted fraction by another FDIV instruction.

### Condition Codes and Boolean Formulas:

	S	X	H	I	N	Z	V	C
	-	-	-	-	-	Δ	Δ	Δ
Z:	Set if quotient is \$0000; cleared otherwise.							
V:	1 if X ≤ D Set if the denominator was less than or equal to the numerator; cleared otherwise.							
C:	X15 • X14 • X13 • X12 • ... • X3 • X2 • X1 • X0 Set if denominator was \$0000; cleared otherwise.							

### Addressing Modes, Machine Code, and Execution Times:

Source Form	Address Mode	Object Code	Cycles	Access Detail
FDIV	INH	18 11	12	0fffffff0

# IBEQ

Increment and Branch if Equal  
to Zero

# IBEQ

**Operation:** (Counter) + 1  $\Rightarrow$  Counter  
If (Counter) = 0, then (PC) + \$0003 + Rel  $\Rightarrow$  PC,

**Description:** Add one to the specified counter register A, B, D, X, Y, or SP. If the counter register has reached zero, branch to the specified relative destination. The IBEQ instruction is encoded into three bytes of machine code including a 9-bit relative offset (–256 to +255 locations from the start of the next instruction).

DBEQ and TBEQ instructions are similar to IBEQ except that the counter is decremented or tested rather than being incremented. Bits 7 and 6 of the instruction postbyte are used to determine which operation is to be performed.

**Condition Codes and Boolean Formulas:**

S	X	H	I	N	Z	V	C
–	–	–	–	–	–	–	–

None affected.

**Addressing Modes, Machine Code, and Execution Times:**

Source Form	Address Mode	Object Code <sup>1</sup>	Cycles	Access Detail
IBEQ <i>abdxys, rel9</i>	REL	04 1b rr	3/3	PPP

Notes:

1. Encoding for 1b is summarized in the following table. Bit 3 is not used (don't care), bit 5 selects branch on zero (IBEQ – 0) or not zero (IBNE – 1) versions, and bit 0 is the sign bit of the 9-bit relative offset. Bits 7 and 6 should be 1:0 for IBEQ.

Count Register	Bits 2:0	Source Form	Object Code (if offset is positive)	Object Code (if offset is negative)
A	000	IBEQ A, <i>rel9</i>	04 80 rr	04 90 rr
B	001	IBEQ B, <i>rel9</i>	04 81 rr	04 91 rr
D	100	IBEQ D, <i>rel9</i>	04 84 rr	04 94 rr
X	101	IBEQ X, <i>rel9</i>	04 85 rr	04 95 rr
Y	110	IBEQ Y, <i>rel9</i>	04 86 rr	04 96 rr
SP	111	IBEQ SP, <i>rel9</i>	04 87 rr	04 97 rr

# IBNE

Increment and Branch if Not  
Equal to Zero

# IBNE

**Operation:** (Counter) + 1  $\Rightarrow$  Counter  
If (Counter) not = 0, then (PC) + \$0003 + Rel  $\Rightarrow$  PC

**Description:** Add one to the specified counter register A, B, D, X, Y, or SP. If the counter register has not been incremented to zero, branch to the specified relative destination. The IBNE instruction is encoded into three bytes of machine code including a 9-bit relative offset (–256 to +255 locations from the start of the next instruction).

DBNE and TBNE instructions are similar to IBNE except that the counter is decremented or tested rather than being incremented. Bits 7 and 6 of the instruction postbyte are used to determine which operation is to be performed.

## Condition Codes and Boolean Formulas:

S	X	H	I	N	Z	V	C
–	–	–	–	–	–	–	–

None affected.

## Addressing Modes, Machine Code, and Execution Times:

Source Form	Address Mode	Object Code <sup>1</sup>	Cycles	Access Detail
IBNE <i>abdxys, rel9</i>	REL	04 1b rr	3/3	PPP

Notes:

- Encoding for 1b is summarized in the following table. Bit 3 is not used (don't care), bit 5 selects branch on zero (IBEQ – 0) or not zero (IBNE – 1) versions, and bit 0 is the sign bit of the 9-bit relative offset. Bits 7 and 6 should be 1:0 for IBNE.

Count Register	Bits 2:0	Source Form	Object Code (if offset is positive)	Object Code (if offset is negative)
A	000	IBNE A, <i>rel9</i>	04 A0 rr	04 B0 rr
B	001	IBNE B, <i>rel9</i>	04 A1 rr	04 B1 rr
D	100	IBNE D, <i>rel9</i>	04 A4 rr	04 B4 rr
X	101	IBNE X, <i>rel9</i>	04 A5 rr	04 B5 rr
Y	110	IBNE Y, <i>rel9</i>	04 A6 rr	04 B6 rr
SP	111	IBNE SP, <i>rel9</i>	04 A7 rr	04 B7 rr

IDIV

Integer Divide

IDIV

**Operation:** (D) ÷ (X) ⇒ X; Remainder ⇒ D

**Description:** Divides an unsigned 16-bit dividend in double accumulator D by an unsigned 16-bit divisor in index register X, producing an unsigned 16-bit quotient in X, and an unsigned 16-bit remainder in D. If both the divisor and the dividend are assumed to have radix points in the same positions, the radix point of the quotient is to the right of bit zero. In the case of division by zero, the quotient is set to \$FFFF, and the remainder is indeterminate.

**Condition Codes and Boolean Formulas:**

S	X	H	I	N	Z	V	C
—	—	—	—	—	Δ	0	Δ

- Z: Set if quotient is \$0000; cleared otherwise.
- V: 0; Cleared.
- C:  $\overline{X15} \bullet \overline{X14} \bullet \overline{X13} \bullet \overline{X12} \bullet \dots \bullet \overline{X3} \bullet \overline{X2} \bullet \overline{X1} \bullet \overline{X0}$   
Set if denominator was \$0000; cleared otherwise.

**Addressing Modes, Machine Code, and Execution Times:**

Source Form	Address Mode	Object Code	Cycles	Access Detail
IDIV	INH	18 10	12	0FFFFFFF0

# IDIVS

## Integer Divide (Signed)

# IDIVS

**Operation:** (D) ÷ (X) ⇒ X; Remainder ⇒ D

**Description:** Performs signed integer division of a signed 16-bit numerator in double accumulator D by a signed 16-bit denominator in index register X, producing a signed 16-bit quotient in X, and a signed 16-bit remainder in D. If division by zero is attempted, the values in D and X are not changed, but the values of the N, Z, and V status bits are undefined.

Other than division by zero, which is not legal and causes the C status bit to be set, the only overflow case is:

$$\frac{\$8000}{\$FFFF} = \frac{-32,768}{-1} = +32,768$$

But the highest positive value that can be represented in a 16-bit two's complement number is 32,767 (\$7FFFF).

### Condition Codes and Boolean Formulas:

S	X	H	I	N	Z	V	C
—	—	—	—	Δ	Δ	Δ	Δ

N: Set if MSB of result is set; cleared otherwise. Undefined after overflow or division by zero.

Z: Set if quotient is \$0000; cleared otherwise. Undefined after overflow or division by zero.

V: Set if the result was > \$7FFF or < \$8000; cleared otherwise. Undefined after division by zero.

C:  $X_{15} \cdot X_{14} \cdot X_{13} \cdot X_{12} \cdot \dots \cdot X_3 \cdot X_2 \cdot X_1 \cdot X_0$   
Set if denominator was \$0000; cleared otherwise.

### Addressing Modes, Machine Code, and Execution Times:

Source Form	Address Mode	Object Code	Cycles	Access Detail
IDIVS	INH	18 15	12	0ffffffefffo

# INC

## Increment Memory

# INC

**Operation:** (M) + \$01  $\Rightarrow$  M

**Description:** Add one to the content of memory location M.

The N, Z and V status bits are set or cleared according to the results of the operation. The C status bit is not affected by the operation, thus allowing the INC instruction to be used as a loop counter in multiple-precision computations.

When operating on unsigned values, only BEQ, BNE, LBEQ, and LBNE branches can be expected to perform consistently. When operating on two's complement values, all signed branches are available.

**Condition Codes and Boolean Formulas:**

S	X	H	I	N	Z	V	C
—	—	—	—	$\Delta$	$\Delta$	$\Delta$	—

- N: Set if MSB of result is set; cleared otherwise.  
Z: Set if result is \$00; cleared otherwise.  
V: Set if there is a two's complement overflow as a result of the operation; cleared otherwise. Two's complement overflow occurs if and only if (M) was \$7F before the operation.

**Addressing Modes, Machine Code, and Execution Times:**

Source Form	Address Mode	Object Code	Cycles	Access Detail
INC <i>opr16a</i>	EXT	72 hh ll	4	rOPw
INC <i>opr0_xysp</i>	IDX	62 xb	3	rPw
INC <i>opr9_xysp</i>	IDX1	62 xb ff	4	rPOw
INC <i>opr16_xysp</i>	IDX2	62 xb ee ff	5	frPPw
INC [D, <i>xysp</i> ]	[D,IDX]	62 xb	6	fIfPrPw
INC [ <i>opr16_xysp</i> ]	[IDX2]	62 xb ee ff	6	fIPrPw

# INCA

## Increment A

# INCA

**Operation:** (A) + \$01  $\Rightarrow$  A

**Description:** Add one to the content of accumulator A.

The N, Z and V status bits are set or cleared according to the results of the operation. The C status bit is not affected by the operation, thus allowing the INC instruction to be used as a loop counter in multiple-precision computations.

When operating on unsigned values, only BEQ, BNE, LBEQ, and LBNE branches can be expected to perform consistently. When operating on two's complement values, all signed branches are available.

### Condition Codes and Boolean Formulas:

S	X	H	I	N	Z	V	C
-	-	-	-	$\Delta$	$\Delta$	$\Delta$	-

- N: Set if MSB of result is set; cleared otherwise.
- Z: Set if result is \$00; cleared otherwise.
- V: Set if there is a two's complement overflow as a result of the operation; cleared otherwise. Two's complement overflow occurs if and only if (A) was \$7F before the operation.

### Addressing Modes, Machine Code, and Execution Times:

Source Form	Address Mode	Object Code	Cycles	Access Detail
INCA	INH	42	1	0



# INCB

Increment B

# INCB

**Operation:** (B) + \$01  $\Rightarrow$  B

**Description:** Add one to the content of accumulator B.

The N, Z and V status bits are set or cleared according to the results of the operation. The C status bit is not affected by the operation, thus allowing the INC instruction to be used as a loop counter in multiple-precision computations.

When operating on unsigned values, only BEQ, BNE, LBEQ, and LBNE branches can be expected to perform consistently. When operating on two's complement values, all signed branches are available.

**Condition Codes and Boolean Formulas:**

S	X	H	I	N	Z	V	C
—	—	—	—	$\Delta$	$\Delta$	$\Delta$	—

- N: Set if MSB of result is set; cleared otherwise.  
Z: Set if result is \$00; cleared otherwise.  
V: Set if there is a two's complement overflow as a result of the operation; cleared otherwise. Two's complement overflow occurs if and only if (B) was \$7F before the operation.

**Addressing Modes, Machine Code, and Execution Times:**

Source Form	Address Mode	Object Code	Cycles	Access Detail
INCB	INH	52	1	0

# INS

## Increment Stack Pointer

# INS

**Operation:** (SP) + \$0001 ⇒ SP

**Description:** Add one to the SP. This instruction is assembled to LEAS 1,SP. The LEAS instruction does not affect condition codes as an INX or INY instruction would.

### Condition Codes and Boolean Formulas:

S	X	H	I	N	Z	V	C
-	-	-	-	-	-	-	-

None affected.

### Addressing Modes, Machine Code, and Execution Times:

Source Form	Address Mode	Object Code	Cycles	Access Detail
INS translates to... LEAS 1,SP	IDX	1B 81	2	pp1

**Notes:**

1. Due to internal CPU requirements, the program word fetch is performed twice to the same address during this instruction.

# INX

## Increment Index Register X

# INX

**Operation:** (X) + \$0001  $\Rightarrow$  X

**Description:** Add one to index register X. LEAX 1,X can produce the same result but LEAX does not affect the Z status bit. Although the LEAX instruction is more flexible, INX requires only one byte of object code.

INX operation affects only the Z status bit.

**Condition Codes and Boolean Formulas:**

S	X	H	I	N	Z	V	C
-	-	-	-	-	$\Delta$	-	-

Z: Set if result is \$0000; cleared otherwise.

**Addressing Modes, Machine Code, and Execution Times:**

Source Form	Address Mode	Object Code	Cycles	Access Detail
INX	INH	08	1	0

# INY

## Increment Index Register Y

# INY

**Operation:**  $(Y) + \$0001 \Rightarrow Y$

**Description:** Add one to index register Y. LEAY 1,Y can produce the same result but LEAY does not affect the Z status bit. Although the LEAY instruction is more flexible, INY requires only one byte of object code.

INY operation affects only the Z status bit.

### Condition Codes and Boolean Formulas:

S	X	H	I	N	Z	V	C
-	-	-	-	-	$\Delta$	-	-

Z: Set if result is \$0000; cleared otherwise.

### Addressing Modes, Machine Code, and Execution Times:

Source Form	Address Mode	Object Code	Cycles	Access Detail
INY	INH	02	1	0

# JMP

Jump

# JMP

**Operation:** Effective Address  $\Rightarrow$  PC

**Description:** Jumps to the instruction stored at the effective address. The effective address is obtained according to the rules for extended or indexed addressing.

**Condition Codes and Boolean Formulas:**

S	X	H	I	N	Z	V	C
—	—	—	—	—	—	—	—

None affected.

**Addressing Modes, Machine Code, and Execution Times:**

Source Form	Address Mode	Object Code	Cycles	Access Detail
JMP <i>opr16a</i>	EXT	06 hh ll	3	PPP
JMP <i>opr0_xysp</i>	IDX	05 xb	3	PPP
JMP <i>opr9_xysp</i>	IDX1	05 xb ff	3	PPP
JMP <i>opr16_xysp</i>	IDX2	05 xb ee ff	4	fPPP
JMP [D, <i>xysp</i> ]	[D,IDX]	05 xb	6	fIfPPP
JMP [ <i>opr16</i> , <i>xysp</i> ]	[IDX2]	05 xb ee ff	6	fIfPPP

# JSR

## Jump to Subroutine

# JSR

**Operation:** (SP) – \$0002  $\Rightarrow$  SP  
RTN<sub>H</sub> : RTN<sub>L</sub>  $\Rightarrow$  M<sub>(SP)</sub> : M<sub>(SP + 1)</sub>  
Subroutine Address  $\Rightarrow$  PC

**Description:** Sets up conditions to return to normal program flow, then transfers control to a subroutine. Uses the address of the instruction following the JSR as a return address.

Decrements the SP by two, to allow the two bytes of the return address to be stacked.

Stacks the return address (the SP points to the high order byte of the return address).

Calculates an effective address according to the rules for extended, direct or indexed addressing.

Jumps to the location determined by the effective address.

Subroutines are normally terminated with an RTS instruction, which restores the return address from the stack.

### Condition Codes and Boolean Formulas:

S	X	H	I	N	Z	V	C
–	–	–	–	–	–	–	–

None affected.

### Addressing Modes, Machine Code, and Execution Times:

Source Form	Address Mode	Object Code	Cycles	Access Detail
JSR <i>opr8a</i>	DIR	17 dd	4	PPPS
JSR <i>opr16a</i>	EXT	16 hh ll	4	PPPS
JSR <i>opr0,xysp</i>	IDX	15 xb	4	PPPS
JSR <i>opr9,xysp</i>	IDX1	15 xb ff	4	PPPS
JSR <i>opr16,xysp</i>	IDX2	15 xb ee ff	5	fPPPS
JSR [D, <i>xysp</i> ]	[D,IDX]	15 xb	7	fifPPPS
JSR [ <i>opr16,xysp</i> ]	[IDX2]	15 xb ee ff	7	fifPPPS

# LBCC

Long Branch if Carry Cleared  
(Same as LBHS)

# LBCC

**Operation:** If  $C = 0$ , then  $(PC) + \$0004 + Rel \Rightarrow PC$

Simple branch

**Description:** Tests the C status bit and branches if  $C = 0$ .

See **3.7 Relative Addressing Mode** for details of branch execution.

**Condition Codes and Boolean Formulas:**

S	X	H	I	N	Z	V	C
—	—	—	—	—	—	—	—

None affected.

**Addressing Modes, Machine Code, and Execution Times:**

Source Form	Address Mode	Object Code	Cycles	Access Detail
LBCC <i>rel16</i>	REL	18 24 qq rr	4/3	OPPP/OPO <sup>1</sup>

Notes:

1. OPPP/OPO indicates this instruction takes four cycles to refill the instruction queue if the branch is taken and three cycles if the branch is not taken.

Branch				Complementary Branch			
Test	Mnemonic	Opcode	Boolean	Test	Mnemonic	Opcode	Comment
$r > m$	LBGT	18 2E	$Z + (N \oplus V) = 0$	$r \leq m$	LBLE	18 2F	Signed
$r \geq m$	LBGE	18 2C	$N \oplus V = 0$	$r < m$	LBLT	18 2D	Signed
$r = m$	LBEQ	18 27	$Z = 1$	$r \neq m$	LBNE	18 26	Signed
$r \leq m$	LBLE	18 2F	$Z + (N \oplus V) = 1$	$r > m$	LBGT	18 2E	Signed
$r < m$	LBLT	18 2D	$N \oplus V = 1$	$r \geq m$	LBGE	18 2C	Signed
$r > m$	LBHI	18 22	$C + Z = 0$	$r \leq m$	LBLS	18 23	Unsigned
$r \geq m$	LBHS/LBCC	18 24	$C = 0$	$r < m$	LBLO/LBCS	18 25	Unsigned
$r = m$	LBEQ	18 27	$Z = 1$	$r \neq m$	LBNE	18 26	Unsigned
$r \leq m$	LBLS	18 23	$C + Z = 1$	$r > m$	LBHI	18 22	Unsigned
$r < m$	LBLO/LBCS	18 25	$C = 1$	$r \geq m$	LBHS/LBCC	18 24	Unsigned
Carry	LBCS	18 25	$C = 1$	No Carry	LBCC	18 24	Simple
Negative	LBMI	18 2B	$N = 1$	Plus	LBPL	18 2A	Simple
Overflow	LBVS	18 29	$V = 1$	No Overflow	LBVC	18 28	Simple
$r = 0$	LBEQ	18 27	$Z = 1$	$r \neq 0$	LBNE	18 26	Simple
Always	LBRA	18 20	—	Never	LBRN	18 21	Unconditional

# LBCS

Long Branch if Carry Set  
(Same as LBLO)

# LBCS

**Operation:** If  $C = 1$ , then  $(PC) + \$0004 + Rel \Rightarrow PC$

Simple branch

**Description:** Tests the C status bit and branches if  $C = 1$ .

See **3.7 Relative Addressing Mode** for details of branch execution.

**Condition Codes and Boolean Formulas:**

S	X	H	I	N	Z	V	C
-	-	-	-	-	-	-	-

None affected.

**Addressing Modes, Machine Code, and Execution Times:**

Source Form	Address Mode	Object Code	Cycles	Access Detail
LBCS <i>rel16</i>	REL	18 25 qq rr	4/3	OPPP/OPO <sup>1</sup>

Notes:

1. OPPP/OPO indicates this instruction takes four cycles to refill the instruction queue if the branch is taken and three cycles if the branch is not taken.

Branch				Complementary Branch			
Test	Mnemonic	Opcode	Boolean	Test	Mnemonic	Opcode	Comment
$r > m$	LBGT	18 2E	$Z + (N \oplus V) = 0$	$r \leq m$	LBLE	18 2F	Signed
$r \geq m$	LBGE	18 2C	$N \oplus V = 0$	$r < m$	LBLT	18 2D	Signed
$r = m$	LBEQ	18 27	$Z = 1$	$r \neq m$	LBNE	18 26	Signed
$r \leq m$	LBLE	18 2F	$Z + (N \oplus V) = 1$	$r > m$	LBGT	18 2E	Signed
$r < m$	LBLT	18 2D	$N \oplus V = 1$	$r \geq m$	LBGE	18 2C	Signed
$r > m$	LBHI	18 22	$C + Z = 0$	$r \leq m$	LBLS	18 23	Unsigned
$r \geq m$	LBHS/LBCC	18 24	$C = 0$	$r < m$	LBLO/LBCS	18 25	Unsigned
$r = m$	LBEQ	18 27	$Z = 1$	$r \neq m$	LBNE	18 26	Unsigned
$r \leq m$	LBLS	18 23	$C + Z = 1$	$r > m$	LBHI	18 22	Unsigned
$r < m$	LBLO/LBCS	18 25	$C = 1$	$r \geq m$	LBHS/LBCC	18 24	Unsigned
Carry	LBCC	18 25	$C = 1$	No Carry	LBCC	18 24	Simple
Negative	LBMI	18 2B	$N = 1$	Plus	LBPL	18 2A	Simple
Overflow	LBVS	18 29	$V = 1$	No Overflow	LBVC	18 28	Simple
$r = 0$	LBEQ	18 27	$Z = 1$	$r \neq 0$	LBNE	18 26	Simple
Always	LBRA	18 20	—	Never	LBRN	18 21	Unconditional



# LBEQ

Long Branch if Equal

# LBEQ

**Operation:** If  $Z = 1$ ,  $(PC) + \$0004 + Rel \Rightarrow PC$

Simple branch

**Description:** Tests the Z status bit and branches if  $Z = 1$ .

See **3.7 Relative Addressing Mode** for details of branch execution.

**Condition Codes and Boolean Formulas:**

S	X	H	I	N	Z	V	C
—	—	—	—	—	—	—	—

None affected.

**Addressing Modes, Machine Code, and Execution Times:**

Source Form	Address Mode	Object Code	Cycles	Access Detail
LBEQ <i>rel16</i>	REL	18 27 qq rr	4/3	OPPP/OPO <sup>1</sup>

Notes:

1. OPPP/OPO indicates this instruction takes four cycles to refill the instruction queue if the branch is taken and three cycles if the branch is not taken.

Branch				Complementary Branch			
Test	Mnemonic	Opcode	Boolean	Test	Mnemonic	Opcode	Comment
$r > m$	LBGT	18 2E	$Z + (N \oplus V) = 0$	$r \leq m$	LBLE	18 2F	Signed
$r \geq m$	LBGE	18 2C	$N \oplus V = 0$	$r < m$	LBLT	18 2D	Signed
$r = m$	LBEQ	18 27	$Z = 1$	$r \neq m$	LBNE	18 26	Signed
$r \leq m$	LBLE	18 2F	$Z + (N \oplus V) = 1$	$r > m$	LBGT	18 2E	Signed
$r < m$	LBLT	18 2D	$N \oplus V = 1$	$r \geq m$	LBGE	18 2C	Signed
$r > m$	LBHI	18 22	$C + Z = 0$	$r \leq m$	LBLS	18 23	Unsigned
$r \geq m$	LBHS/LBCC	18 24	$C = 0$	$r < m$	LBLO/LBCS	18 25	Unsigned
$r = m$	LBEQ	18 27	$Z = 1$	$r \neq m$	LBNE	18 26	Unsigned
$r \leq m$	LBLS	18 23	$C + Z = 1$	$r > m$	LBHI	18 22	Unsigned
$r < m$	LBLO/LBCS	18 25	$C = 1$	$r \geq m$	LBHS/LBCC	18 24	Unsigned
Carry	LBCS	18 25	$C = 1$	No Carry	LBCC	18 24	Simple
Negative	LBMI	18 2B	$N = 1$	Plus	LBPL	18 2A	Simple
Overflow	LBVS	18 29	$V = 1$	No Overflow	LBVC	18 28	Simple
$r = 0$	LBEQ	18 27	$Z = 1$	$r \neq 0$	LBNE	18 26	Simple
Always	LBRA	18 20	—	Never	LBRN	18 21	Unconditional

# LBGE

Long Branch if Greater Than or Equal to Zero

# LBGE

**Operation:** If  $N \oplus V = 0$ ,  $(PC) + \$0004 + \text{Rel} \Rightarrow PC$

For signed two's complement numbers,  
if (Accumulator)  $\geq$  Memory), then branch

**Description:** If LBGE is executed immediately after execution of CBA, CMPA, CMPB, CMPD, CPX, CPY, SBA, SUBA, SUBB, or SUBD, a branch occurs if and only if the two's complement number in the accumulator was greater than or equal to the two's complement number in memory.

See **3.7 Relative Addressing Mode** for details of branch execution.

## Condition Codes and Boolean Formulas:

S	X	H	I	N	Z	V	C
-	-	-	-	-	-	-	-

None affected.

## Addressing Modes, Machine Code, and Execution Times:

Source Form	Address Mode	Object Code	Cycles	Access Detail
LBGE <i>rel16</i>	REL	18 2C qq rr	4/3	OPPP/OPO <sup>1</sup>

Notes:

1. OPPP/OPO indicates this instruction takes four cycles to refill the instruction queue if the branch is taken and three cycles if the branch is not taken.

Branch				Complementary Branch			
Test	Mnemonic	Opcode	Boolean	Test	Mnemonic	Opcode	Comment
$r > m$	LBGT	18 2E	$Z + (N \oplus V) = 0$	$r \leq m$	LBLE	18 2F	Signed
$r \geq m$	LBGE	18 2C	$N \oplus V = 0$	$r < m$	LBLT	18 2D	Signed
$r = m$	LBEQ	18 27	$Z = 1$	$r \neq m$	LBNE	18 26	Signed
$r \leq m$	LBLE	18 2F	$Z + (N \oplus V) = 1$	$r > m$	LBGT	18 2E	Signed
$r < m$	LBLT	18 2D	$N \oplus V = 1$	$r \geq m$	LBGE	18 2C	Signed
$r > m$	LBHI	18 22	$C + Z = 0$	$r \leq m$	LBLS	18 23	Unsigned
$r \geq m$	LBHS/LBCC	18 24	$C = 0$	$r < m$	LBLO/LBCS	18 25	Unsigned
$r = m$	LBEQ	18 27	$Z = 1$	$r \neq m$	LBNE	18 26	Unsigned
$r \leq m$	LBLS	18 23	$C + Z = 1$	$r > m$	LBHI	18 22	Unsigned
$r < m$	LBLO/LBCS	18 25	$C = 1$	$r \geq m$	LBHS/LBCC	18 24	Unsigned
Carry	LBCS	18 25	$C = 1$	No Carry	LBCC	18 24	Simple
Negative	LBMI	18 2B	$N = 1$	Plus	LBPL	18 2A	Simple
Overflow	LBVS	18 29	$V = 1$	No Overflow	LBVC	18 28	Simple
$r = 0$	LBEQ	18 27	$Z = 1$	$r \neq 0$	LBNE	18 26	Simple
Always	LBRA	18 20	—	Never	LBRN	18 21	Unconditional

LBGT

Long Branch if Greater Than Zero

LBGT

**Operation:** If  $Z + (N \oplus V) = 0$ , then  $(PC) + \$0004 + Rel \Rightarrow PC$

For signed two's complement numbers,  
If (Accumulator) > (Memory), then branch

**Description:** If LBGT is executed immediately after execution of CBA, CMPA, CMPB, CMPD, CPX, CPY, SBA, SUBA, SUBB, or SUBD, a branch occurs if and only if the two's complement number in the accumulator was greater than the two's complement number in memory.

See **3.7 Relative Addressing Mode** for details of branch execution.

**Condition Codes and Boolean Formulas:**

S	X	H	I	N	Z	V	C
—	—	—	—	—	—	—	—

None affected.

**Addressing Modes, Machine Code, and Execution Times:**

Source Form	Address Mode	Object Code	Cycles	Access Detail
LBGT <i>rel16</i>	REL	18 2E qq rr	4/3	OPPP/OPO <sup>1</sup>

Notes:  
1. OPPP/OPO indicates this instruction takes four cycles to refill the instruction queue if the branch is taken and three cycles if the branch is not taken.

Branch				Complementary Branch			
Test	Mnemonic	Opcode	Boolean	Test	Mnemonic	Opcode	Comment
r>m	LBGT	18 2E	$Z + (N \oplus V) = 0$	r≤m	LBLE	18 2F	Signed
r≥m	LBGE	18 2C	$N \oplus V = 0$	r<m	LBLT	18 2D	Signed
r=m	LBEQ	18 27	$Z = 1$	r≠m	LBNE	18 26	Signed
r≤m	LBLE	18 2F	$Z + (N \oplus V) = 1$	r>m	LBGT	18 2E	Signed
r<m	LBLT	18 2D	$N \oplus V = 1$	r≥m	LBGE	18 2C	Signed
r>m	LBHI	18 22	$C + Z = 0$	r≤m	LBLS	18 23	Unsigned
r≥m	LBHS/LBCC	18 24	$C = 0$	r<m	LBLO/LBCS	18 25	Unsigned
r=m	LBEQ	18 27	$Z = 1$	r≠m	LBNE	18 26	Unsigned
r≤m	LBLS	18 23	$C + Z = 1$	r>m	LBHI	18 22	Unsigned
r<m	LBLO/LBCS	18 25	$C = 1$	r≥m	LBHS/LBCC	18 24	Unsigned
Carry	LBCS	18 25	$C = 1$	No Carry	LBCC	18 24	Simple
Negative	LBMI	18 2B	$N = 1$	Plus	LBPL	18 2A	Simple
Overflow	LBVS	18 29	$V = 1$	No Overflow	LBVC	18 28	Simple
r=0	LBEQ	18 27	$Z = 1$	r≠0	LBNE	18 26	Simple
Always	LBRA	18 20	—	Never	LBRN	18 21	Unconditional

# LBHI

## Long Branch if Higher

# LBHI

**Operation:** If  $C + Z = 0$ , then  $(PC) + \$0004 + Rel \Rightarrow PC$

For unsigned binary numbers, if  $(\text{Accumulator}) > (\text{Memory})$ , then branch

**Description:** If LBHI is executed immediately after execution of CBA, CMPA, CMPB, CMPD, CPX, CPY, SBA, SUBA, SUBB, or SUBD, a branch occurs if and only if the unsigned binary number in the accumulator was greater than the unsigned binary number in memory. This instruction is generally not useful after INC/DEC, LD/ST, TST/CLR/COM because these instructions do not affect the C status bit.

See 3.7 Relative Addressing Mode for details of branch execution.

### Condition Codes and Boolean Formulas:

S	X	H	I	N	Z	V	C
-	-	-	-	-	-	-	-

None affected.

### Addressing Modes, Machine Code, and Execution Times:

Source Form	Address Mode	Object Code	Cycles	Access Detail
LBHI rel16	REL	18 22 qq rr	4/3	OPPP/OPO <sup>1</sup>

Notes:

1. OPPP/OPO indicates this instruction takes four cycles to refill the instruction queue if the branch is taken and three cycles if the branch is not taken.

Branch				Complementary Branch			
Test	Mnemonic	Opcode	Boolean	Test	Mnemonic	Opcode	Comment
$r > m$	LBGT	18 2E	$Z + (N \oplus V) = 0$	$r \leq m$	LBLE	18 2F	Signed
$r \geq m$	LBGE	18 2C	$N \oplus V = 0$	$r < m$	LBLT	18 2D	Signed
$r = m$	LBEQ	18 27	$Z = 1$	$r \neq m$	LBNE	18 26	Signed
$r \leq m$	LBLE	18 2F	$Z + (N \oplus V) = 1$	$r > m$	LBGT	18 2E	Signed
$r < m$	LBLT	18 2D	$N \oplus V = 1$	$r \geq m$	LBGE	18 2C	Signed
$r > m$	LBHI	18 22	$C + Z = 0$	$r \leq m$	LBLS	18 23	Unsigned
$r \geq m$	LBHS/LBCC	18 24	$C = 0$	$r < m$	LBLO/LBCS	18 25	Unsigned
$r = m$	LBEQ	18 27	$Z = 1$	$r \neq m$	LBNE	18 26	Unsigned
$r \leq m$	LBLS	18 23	$C + Z = 1$	$r > m$	LBHI	18 22	Unsigned
$r < m$	LBLO/LBCS	18 25	$C = 1$	$r \geq m$	LBHS/LBCC	18 24	Unsigned
Carry	LBCS	18 25	$C = 1$	No Carry	LBCC	18 24	Simple
Negative	LBMI	18 2B	$N = 1$	Plus	LBPL	18 2A	Simple
Overflow	LBVS	18 29	$V = 1$	No Overflow	LBVC	18 28	Simple
$r = 0$	LBEQ	18 27	$Z = 1$	$r \neq 0$	LBNE	18 26	Simple
Always	LBRA	18 20	—	Never	LBRN	18 21	Unconditional

LBHS

Long Branch if Higher or Same  
(Same as LBCC)

LBHS

**Operation:** If C = 0, then (PC) + \$0004 + Rel  $\Rightarrow$  PC

For unsigned binary numbers, if (Accumulator)  $\geq$  (Memory), then branch

**Description:** If LBHS is executed immediately after execution of CBA, CMPA, CMPB, CMPD, CPX, CPY, SBA, SUBA, SUBB, or SUBD, a branch occurs if and only if the unsigned binary number in the accumulator was greater than or equal to the unsigned binary number in memory. This instruction is generally not useful after INC/DEC, LD/ST, TST/CLR/COM because these instructions do not affect the C status bit.

See **3.7 Relative Addressing Mode** for details of branch execution.

Condition Codes and Boolean Formulas:

S	X	H	I	N	Z	V	C
—	—	—	—	—	—	—	—

None affected.

Addressing Modes, Machine Code, and Execution Times:

Source Form	Address Mode	Object Code	Cycles	Access Detail
LBHS <i>rel16</i>	REL	18 24 qq rr	4/3	OPPP/OPO <sup>1</sup>

Notes:

1. OPPP/OPO indicates this instruction takes four cycles to refill the instruction queue if the branch is taken and three cycles if the branch is not taken.

Branch				Complementary Branch			
Test	Mnemonic	Opcode	Boolean	Test	Mnemonic	Opcode	Comment
r>m	LBGT	18 2E	Z + (N $\oplus$ V) = 0	r≤m	LBLE	18 2F	Signed
r≥m	LBGE	18 2C	N $\oplus$ V = 0	r<m	LBLT	18 2D	Signed
r=m	LBEQ	18 27	Z = 1	r≠m	LBNE	18 26	Signed
r≤m	LBLE	18 2F	Z + (N $\oplus$ V) = 1	r>m	LBGT	18 2E	Signed
r<m	LBLT	18 2D	N $\oplus$ V = 1	r≥m	LBGE	18 2C	Signed
r>m	LBHI	18 22	C + Z = 0	r≤m	LBLS	18 23	Unsigned
r≥m	LBHS/LBCC	18 24	C = 0	r<m	LBLO/LBCS	18 25	Unsigned
r=m	LBEQ	18 27	Z = 1	r≠m	LBNE	18 26	Unsigned
r≤m	LBLS	18 23	C + Z = 1	r>m	LBHI	18 22	Unsigned
r<m	LBLO/LBCS	18 25	C = 1	r≥m	LBHS/LBCC	18 24	Unsigned
Carry	LBCS	18 25	C = 1	No Carry	LBCC	18 24	Simple
Negative	LBMI	18 2B	N = 1	Plus	LBPL	18 2A	Simple
Overflow	LBVS	18 29	V = 1	No Overflow	LBVC	18 28	Simple
r=0	LBEQ	18 27	Z = 1	r≠0	LBNE	18 26	Simple
Always	LBRA	18 20	—	Never	LBRN	18 21	Unconditional

# LBLE

## Long Branch if Less Than or Equal to Zero

# LBLE

**Operation:** If  $Z + (N \oplus V) = 1$ , then  $(PC) + \$0004 + \text{Rel} \Rightarrow PC$

For signed two's complement numbers,  
if  $(\text{Accumulator}) \leq (\text{Memory})$ , then branch

**Description:** If LBLE is executed immediately after execution of CBA, CMPA, CMPB, CMPD, CPX, CPY, SBA, SUBA, SUBB, or SUBD, a branch occurs if and only if the two's complement number in the accumulator was less than or equal to the two's complement number in memory.

See **3.7 Relative Addressing Mode** for details of branch execution.

### Condition Codes and Boolean Formulas:

S	X	H	I	N	Z	V	C
-	-	-	-	-	-	-	-

None affected.

### Addressing Modes, Machine Code, and Execution Times:

Source Form	Address Mode	Object Code	Cycles	Access Detail
LBLE <i>rel16</i>	REL	18 2F qq rr	4/3	OPPP/OPO <sup>1</sup>

Notes:

1. OPPP/OPO indicates this instruction takes four cycles to refill the instruction queue if the branch is taken and three cycles if the branch is not taken.

Branch				Complementary Branch			
Test	Mnemonic	Opcode	Boolean	Test	Mnemonic	Opcode	Comment
$r > m$	LBGT	18 2E	$Z + (N \oplus V) = 0$	$r \leq m$	LBLE	18 2F	Signed
$r \geq m$	LBGE	18 2C	$N \oplus V = 0$	$r < m$	LBLT	18 2D	Signed
$r = m$	LBEQ	18 27	$Z = 1$	$r \neq m$	LBNE	18 26	Signed
$r \leq m$	LBLE	18 2F	$Z + (N \oplus V) = 1$	$r > m$	LBGT	18 2E	Signed
$r < m$	LBLT	18 2D	$N \oplus V = 1$	$r \geq m$	LBGE	18 2C	Signed
$r > m$	LBHI	18 22	$C + Z = 0$	$r \leq m$	LBLS	18 23	Unsigned
$r \geq m$	LBHS/LBCC	18 24	$C = 0$	$r < m$	LBLO/LBCS	18 25	Unsigned
$r = m$	LBEQ	18 27	$Z = 1$	$r \neq m$	LBNE	18 26	Unsigned
$r \leq m$	LBLS	18 23	$C + Z = 1$	$r > m$	LBHI	18 22	Unsigned
$r < m$	LBLO/LBCS	18 25	$C = 1$	$r \geq m$	LBHS/LBCC	18 24	Unsigned
Carry	LBCS	18 25	$C = 1$	No Carry	LBCC	18 24	Simple
Negative	LBMI	18 2B	$N = 1$	Plus	LBPL	18 2A	Simple
Overflow	LBVS	18 29	$V = 1$	No Overflow	LBVC	18 28	Simple
$r = 0$	LBEQ	18 27	$Z = 1$	$r \neq 0$	LBNE	18 26	Simple
Always	LBRA	18 20	—	Never	LBRN	18 21	Unconditional

# LBLO

Long Branch if Lower  
(Same as LBCS)

# LBLO

**Operation:** If C = 1, then (PC) + \$0004 + Rel  $\Rightarrow$  PC

For unsigned binary numbers, if (Accumulator) < (Memory), then branch

**Description:** If LBLO is executed immediately after execution of CBA, CMPA, CMPB, CMPD, CPX, CPY, SBA, SUBA, SUBB, or SUBD, a branch occurs if and only if the unsigned binary number in the accumulator was less than the unsigned binary number in memory. This instruction is generally not useful after INC/DEC, LD/ST, TST/CLR/COM because these instructions do not affect the C status bit.

See 3.7 Relative Addressing Mode for details of branch execution.

**Condition Codes and Boolean Formulas:**

S	X	H	I	N	Z	V	C
—	—	—	—	—	—	—	—

None affected.

**Addressing Modes, Machine Code, and Execution Times:**

Source Form	Address Mode	Object Code	Cycles	Access Detail
LBLO <i>rel</i> /16	REL	18 25 qq rr	4/3	OPPP/OPO <sup>1</sup>

Notes:

1. OPPP/OPO indicates this instruction takes four cycles to refill the instruction queue if the branch is taken and three cycles if the branch is not taken.

Branch				Complementary Branch			
Test	Mnemonic	Opcode	Boolean	Test	Mnemonic	Opcode	Comment
r>m	LBGT	18 2E	Z + (N $\oplus$ V) = 0	r≤m	LBLE	18 2F	Signed
r≥m	LBGE	18 2C	N $\oplus$ V = 0	r<m	LBLT	18 2D	Signed
r=m	LBEQ	18 27	Z = 1	r≠m	LBNE	18 26	Signed
r≤m	LBLE	18 2F	Z + (N $\oplus$ V) = 1	r>m	LBGT	18 2E	Signed
r<m	LBLT	18 2D	N $\oplus$ V = 1	r≥m	LBGE	18 2C	Signed
r>m	LBHI	18 22	C + Z = 0	r≤m	LBLS	18 23	Unsigned
r≥m	LBHS/LBCC	18 24	C = 0	r<m	LBLO/LBCS	18 25	Unsigned
r=m	LBEQ	18 27	Z = 1	r≠m	LBNE	18 26	Unsigned
r≤m	LBLS	18 23	C + Z = 1	r>m	LBHI	18 22	Unsigned
r<m	LBLO/LBCS	18 25	C = 1	r≥m	LBHS/LBCC	18 24	Unsigned
Carry	LBCS	18 25	C = 1	No Carry	LBCC	18 24	Simple
Negative	LBMI	18 2B	N = 1	Plus	LBPL	18 2A	Simple
Overflow	LBVS	18 29	V = 1	No Overflow	LBVC	18 28	Simple
r=0	LBEQ	18 27	Z = 1	r≠0	LBNE	18 26	Simple
Always	LBRA	18 20	—	Never	LBRN	18 21	Unconditional

# LBLS

Long Branch if Lower or Same

# LBLS

**Operation:** If  $C + Z = 1$ , then  $(PC) + \$0004 + Rel \Rightarrow PC$

For unsigned binary numbers, if  $(\text{Accumulator}) \leq (\text{Memory})$ , then branch

**Description:** If LBLS is executed immediately after execution of CBA, CMPA, CMPB, CMPD, CPX, CPY, SBA, SUBA, SUBB, or SUBD, a branch occurs if and only if the unsigned binary number in the accumulator was less than or equal to the unsigned binary number in memory. This instruction is generally not useful after INC/DEC, LD/ST, TST/CLR/COM because these instructions do not affect the C status bit.

See 3.7 Relative Addressing Mode for details of branch execution.

## Condition Codes and Boolean Formulas:

S	X	H	I	N	Z	V	C
—	—	—	—	—	—	—	—

None affected.

## Addressing Modes, Machine Code, and Execution Times:

Source Form	Address Mode	Object Code	Cycles	Access Detail
LBLS <i>rel16</i>	REL	18 23 qq rr	4/3	OPPP/OPO <sup>1</sup>

Notes:

1. OPPP/OPO indicates this instruction takes four cycles to refill the instruction queue if the branch is taken and three cycles if the branch is not taken.

Branch				Complementary Branch			
Test	Mnemonic	Opcode	Boolean	Test	Mnemonic	Opcode	Comment
$r > m$	LBGT	18 2E	$Z + (N \oplus V) = 0$	$r \leq m$	LBLE	18 2F	Signed
$r \geq m$	LBGE	18 2C	$N \oplus V = 0$	$r < m$	LBLT	18 2D	Signed
$r = m$	LBEQ	18 27	$Z = 1$	$r \neq m$	LBNE	18 26	Signed
$r \leq m$	LBLE	18 2F	$Z + (N \oplus V) = 1$	$r > m$	LBGT	18 2E	Signed
$r < m$	LBLT	18 2D	$N \oplus V = 1$	$r \geq m$	LBGE	18 2C	Signed
$r > m$	LBHI	18 22	$C + Z = 0$	$r \leq m$	LBLS	18 23	Unsigned
$r \geq m$	LBHS/LBCC	18 24	$C = 0$	$r < m$	LBLO/LBCS	18 25	Unsigned
$r = m$	LBEQ	18 27	$Z = 1$	$r \neq m$	LBNE	18 26	Unsigned
$r \leq m$	LBLS	18 23	$C + Z = 1$	$r > m$	LBHI	18 22	Unsigned
$r < m$	LBLO/LBCS	18 25	$C = 1$	$r \geq m$	LBHS/LBCC	18 24	Unsigned
Carry	LBCS	18 25	$C = 1$	No Carry	LBCC	18 24	Simple
Negative	LBMI	18 2B	$N = 1$	Plus	LBPL	18 2A	Simple
Overflow	LBVS	18 29	$V = 1$	No Overflow	LBVC	18 28	Simple
$r = 0$	LBEQ	18 27	$Z = 1$	$r \neq 0$	LBNE	18 26	Simple
Always	LBRA	18 20	—	Never	LBRN	18 21	Unconditional



# LBLT

Long Branch if Less Than Zero

# LBLT

**Operation:** If  $N \oplus V = 1$ ,  $(PC) + \$0004 + Rel \Rightarrow PC$

For signed two's complement numbers,  
if (Accumulator) < (Memory), then branch

**Description:** If LBLT is executed immediately after execution of CBA, CMPA, CMPB, CMPD, CPX, CPY, SBA, SUBA, SUBB, or SUBD, a branch occurs if and only if the two's complement number in the accumulator was less than the two's complement number in memory.

See **3.7 Relative Addressing Mode** for details of branch execution.

**Condition Codes and Boolean Formulas:**

S	X	H	I	N	Z	V	C
—	—	—	—	—	—	—	—

None affected.

**Addressing Modes, Machine Code, and Execution Times:**

Source Form	Address Mode	Object Code	Cycles	Access Detail
LBLT <i>rel16</i>	REL	18 2D qq rr	4/3	OPPP/OPO <sup>1</sup>

Notes:

1. OPPP/OPO indicates this instruction takes four cycles to refill the instruction queue if the branch is taken and three cycles if the branch is not taken.

Branch				Complementary Branch			
Test	Mnemonic	Opcode	Boolean	Test	Mnemonic	Opcode	Comment
$r > m$	LBGT	18 2E	$Z + (N \oplus V) = 0$	$r \leq m$	LBLE	18 2F	Signed
$r \geq m$	LBGE	18 2C	$N \oplus V = 0$	$r < m$	LBLT	18 2D	Signed
$r = m$	LBEQ	18 27	$Z = 1$	$r \neq m$	LBNE	18 26	Signed
$r \leq m$	LBLE	18 2F	$Z + (N \oplus V) = 1$	$r > m$	LBGT	18 2E	Signed
$r < m$	LBLT	18 2D	$N \oplus V = 1$	$r \geq m$	LBGE	18 2C	Signed
$r > m$	LBHI	18 22	$C + Z = 0$	$r \leq m$	LBLS	18 23	Unsigned
$r \geq m$	LBHS/LBCC	18 24	$C = 0$	$r < m$	LBLO/LBCS	18 25	Unsigned
$r = m$	LBEQ	18 27	$Z = 1$	$r \neq m$	LBNE	18 26	Unsigned
$r \leq m$	LBLS	18 23	$C + Z = 1$	$r > m$	LBHI	18 22	Unsigned
$r < m$	LBLO/LBCS	18 25	$C = 1$	$r \geq m$	LBHS/LBCC	18 24	Unsigned
Carry	LBCS	18 25	$C = 1$	No Carry	LBCC	18 24	Simple
Negative	LBMI	18 2B	$N = 1$	Plus	LBPL	18 2A	Simple
Overflow	LBVS	18 29	$V = 1$	No Overflow	LBVC	18 28	Simple
$r = 0$	LBEQ	18 27	$Z = 1$	$r \neq 0$	LBNE	18 26	Simple
Always	LBRA	18 20	—	Never	LBRN	18 21	Unconditional

# LBMI

## Long Branch if Minus

# LBMI

**Operation:** If  $N = 1$ , then  $(PC) + \$0004 + Rel \Rightarrow PC$

Simple branch

**Description:** Tests the N status bit and branches if  $N = 1$ .

See **3.7 Relative Addressing Mode** for details of branch execution.

**Condition Codes and Boolean Formulas:**

S	X	H	I	N	Z	V	C
-	-	-	-	-	-	-	-

None affected.

**Addressing Modes, Machine Code, and Execution Times:**

Source Form	Address Mode	Object Code	Cycles	Access Detail
LBMI <i>rel16</i>	REL	18 2B qq rr	4/3	OPPP/OPO <sup>1</sup>

Notes:

1. OPPP/OPO indicates this instruction takes four cycles to refill the instruction queue if the branch is taken and three cycles if the branch is not taken.

Branch				Complementary Branch			
Test	Mnemonic	Opcode	Boolean	Test	Mnemonic	Opcode	Comment
$r > m$	LBGT	18 2E	$Z + (N \oplus V) = 0$	$r \leq m$	LBLE	18 2F	Signed
$r \geq m$	LBGE	18 2C	$N \oplus V = 0$	$r < m$	LBLT	18 2D	Signed
$r = m$	LBEQ	18 27	$Z = 1$	$r \neq m$	LBNE	18 26	Signed
$r \leq m$	LBLE	18 2F	$Z + (N \oplus V) = 1$	$r > m$	LBGT	18 2E	Signed
$r < m$	LBLT	18 2D	$N \oplus V = 1$	$r \geq m$	LBGE	18 2C	Signed
$r > m$	LBHI	18 22	$C + Z = 0$	$r \leq m$	LBLS	18 23	Unsigned
$r \geq m$	LBHS/LBCC	18 24	$C = 0$	$r < m$	LBLO/LBCS	18 25	Unsigned
$r = m$	LBEQ	18 27	$Z = 1$	$r \neq m$	LBNE	18 26	Unsigned
$r \leq m$	LBLS	18 23	$C + Z = 1$	$r > m$	LBHI	18 22	Unsigned
$r < m$	LBLO/LBCS	18 25	$C = 1$	$r \geq m$	LBHS/LBCC	18 24	Unsigned
Carry	LBCC	18 25	$C = 1$	No Carry	LBCC	18 24	Simple
Negative	LBMI	18 2B	$N = 1$	Plus	LBPL	18 2A	Simple
Overflow	LBVS	18 29	$V = 1$	No Overflow	LBVC	18 28	Simple
$r = 0$	LBEQ	18 27	$Z = 1$	$r \neq 0$	LBNE	18 26	Simple
Always	LBRA	18 20	—	Never	LBRN	18 21	Unconditional

# LBNE

Long Branch if Not Equal to Zero

# LBNE

**Operation:** If  $Z = 0$ , then  $(PC) + \$0004 + Rel \Rightarrow PC$

Simple branch

**Description:** Tests the Z status bit and branches if  $Z = 0$ .

See **3.7 Relative Addressing Mode** for details of branch execution.

**Condition Codes and Boolean Formulas:**

S	X	H	I	N	Z	V	C
—	—	—	—	—	—	—	—

None affected.

**Addressing Modes, Machine Code, and Execution Times:**

Source Form	Address Mode	Object Code	Cycles	Access Detail
LBNE <i>rel16</i>	REL	18 26 qq rr	4/3	OPPP/OPO <sup>1</sup>

Notes:

1. OPPP/OPO indicates this instruction takes four cycles to refill the instruction queue if the branch is taken and three cycles if the branch is not taken.

Branch				Complementary Branch			
Test	Mnemonic	Opcode	Boolean	Test	Mnemonic	Opcode	Comment
$r > m$	LBGT	18 2E	$Z + (N \oplus V) = 0$	$r \leq m$	LBLE	18 2F	Signed
$r \geq m$	LBGE	18 2C	$N \oplus V = 0$	$r < m$	LBLT	18 2D	Signed
$r = m$	LBEQ	18 27	$Z = 1$	$r \neq m$	LBNE	18 26	Signed
$r \leq m$	LBLE	18 2F	$Z + (N \oplus V) = 1$	$r > m$	LBGT	18 2E	Signed
$r < m$	LBLT	18 2D	$N \oplus V = 1$	$r \geq m$	LBGE	18 2C	Signed
$r > m$	LBHI	18 22	$C + Z = 0$	$r \leq m$	LBLS	18 23	Unsigned
$r \geq m$	LBHS/LBCC	18 24	$C = 0$	$r < m$	LBLO/LBCS	18 25	Unsigned
$r = m$	LBEQ	18 27	$Z = 1$	$r \neq m$	LBNE	18 26	Unsigned
$r \leq m$	LBLS	18 23	$C + Z = 1$	$r > m$	LBHI	18 22	Unsigned
$r < m$	LBLO/LBCS	18 25	$C = 1$	$r \geq m$	LBHS/LBCC	18 24	Unsigned
Carry	LBCS	18 25	$C = 1$	No Carry	LBCC	18 24	Simple
Negative	LBMI	18 2B	$N = 1$	Plus	LBPL	18 2A	Simple
Overflow	LBVS	18 29	$V = 1$	No Overflow	LBVC	18 28	Simple
$r = 0$	LBEQ	18 27	$Z = 1$	$r \neq 0$	LBNE	18 26	Simple
Always	LBRA	18 20	—	Never	LBRN	18 21	Unconditional

# LBPL

## Long Branch if Plus

# LBPL

**Operation:** If  $N = 0$ , then  $(PC) + \$0004 + Rel \Rightarrow PC$

Simple branch

**Description:** Tests the N status bit and branches if  $N = 0$ .

See 3.7 Relative Addressing Mode for details of branch execution.

**Condition Codes and Boolean Formulas:**

S	X	H	I	N	Z	V	C
-	-	-	-	-	-	-	-

None affected.

**Addressing Modes, Machine Code, and Execution Times:**

Source Form	Address Mode	Object Code	Cycles	Access Detail
LBPL <i>rel16</i>	REL	18 2A qq rr	4/3	OPPP/OPO <sup>1</sup>

Notes:

1. OPPP/OPO indicates this instruction takes four cycles to refill the instruction queue if the branch is taken and three cycles if the branch is not taken.

Branch				Complementary Branch			
Test	Mnemonic	Opcode	Boolean	Test	Mnemonic	Opcode	Comment
$r > m$	LBGT	18 2E	$Z + (N \oplus V) = 0$	$r \leq m$	LBLE	18 2F	Signed
$r \geq m$	LBGE	18 2C	$N \oplus V = 0$	$r < m$	LBLT	18 2D	Signed
$r = m$	LBEQ	18 27	$Z = 1$	$r \neq m$	LBNE	18 26	Signed
$r \leq m$	LBLE	18 2F	$Z + (N \oplus V) = 1$	$r > m$	LBGT	18 2E	Signed
$r < m$	LBLT	18 2D	$N \oplus V = 1$	$r \geq m$	LBGE	18 2C	Signed
$r > m$	LBHI	18 22	$C + Z = 0$	$r \leq m$	LBLS	18 23	Unsigned
$r \geq m$	LBHS/LBCC	18 24	$C = 0$	$r < m$	LBLO/LBCS	18 25	Unsigned
$r = m$	LBEQ	18 27	$Z = 1$	$r \neq m$	LBNE	18 26	Unsigned
$r \leq m$	LBLS	18 23	$C + Z = 1$	$r > m$	LBHI	18 22	Unsigned
$r < m$	LBLO/LBCS	18 25	$C = 1$	$r \geq m$	LBHS/LBCC	18 24	Unsigned
Carry	LBCS	18 25	$C = 1$	No Carry	LBCC	18 24	Simple
Negative	LBMI	18 2B	$N = 1$	Plus	LBPL	18 2A	Simple
Overflow	LBVS	18 29	$V = 1$	No Overflow	LBVC	18 28	Simple
$r = 0$	LBEQ	18 27	$Z = 1$	$r \neq 0$	LBNE	18 26	Simple
Always	LBRA	18 20	—	Never	LBRN	18 21	Unconditional

LBRA

Long Branch Always

LBRA

**Operation:** (PC) + \$0004 + Rel ⇒ PC

**Description:** Unconditional branch to an address calculated as shown in the expression. Rel is a relative offset stored as a two's complement number in the second and third bytes of machine code corresponding to the long branch instruction.

Execution time is longer when a conditional branch is taken than when it is not, because the instruction queue must be refilled before execution resumes at the new address. Since the LBRA branch condition is always satisfied, the branch is always taken, and the instruction queue must always be refilled, so execution time is always the larger value.

See **3.7 Relative Addressing Mode** for details of branch execution.

**Condition Codes and Boolean Formulas:**

S	X	H	I	N	Z	V	C
-	-	-	-	-	-	-	-

None affected.

**Addressing Modes, Machine Code, and Execution Times:**

Source Form	Address Mode	Object Code	Cycles	Access Detail
LBRA <i>rel16</i>	REL	18 20 qq rr	4	OPPP

## Long Branch Never

# LIBRIN

**Operation:**  $(PC) + \$0004 \Rightarrow PC$

**Description:** Never branches. LBRN is effectively a 4-byte NOP that requires three cycles to execute. LBRN is included in the instruction set to provide a complement to the LBRA instruction. The instruction is useful during program debug, to negate the effect of another branch instruction without disturbing the offset byte. A complement for LBRA is also useful in compiler implementations.

Execution time is longer when a conditional branch is taken than when it is not, because the instruction queue must be refilled before execution resumes at the new address. Since the LBRN branch condition is never satisfied, the branch is never taken, and the queue does not need to be refilled, so execution time is always the smaller value.

### Condition Codes and Boolean Formulas:

S	X	H	I	N	Z	V	C
-	-	-	-	-	-	-	-

None affected.

### Addressing Modes, Machine Code, and Execution Times:

Source Form	Address Mode	Object Code	Cycles	Access Detail
LBRN <i>rel16</i>	REL	18 21 qq rr	3	OPO

# LBVC

Long Branch if Overflow Cleared

# LBVC

**Operation:** If  $V = 0$ , then  $(PC) + \$0004 + Rel \Rightarrow PC$

Simple branch

**Description:** Tests the  $V$  status bit and branches if  $V = 0$ .

LBVC causes a branch when a previous operation on two's complement binary values does not cause an overflow. That is, when LBVC follows a two's complement operation, a branch occurs when the result of the operation is valid.

See **3.7 Relative Addressing Mode** for details of branch execution.

## Condition Codes and Boolean Formulas:

S	X	H	I	N	Z	V	C
—	—	—	—	—	—	—	—

None affected.

## Addressing Modes, Machine Code, and Execution Times:

Source Form	Address Mode	Object Code	Cycles	Access Detail
LBVC <i>rel16</i>	REL	18 28 qq rr	4/3	OPPP/OPO <sup>1</sup>

Notes:

1. OPPP/OPO indicates this instruction takes four cycles to refill the instruction queue if the branch is taken and three cycles if the branch is not taken.

Branch				Complementary Branch			
Test	Mnemonic	Opcode	Boolean	Test	Mnemonic	Opcode	Comment
$r > m$	LBGT	18 2E	$Z + (N \oplus V) = 0$	$r \leq m$	LBLE	18 2F	Signed
$r \geq m$	LBGE	18 2C	$N \oplus V = 0$	$r < m$	LBLT	18 2D	Signed
$r = m$	LBEQ	18 27	$Z = 1$	$r \neq m$	LBNE	18 26	Signed
$r \leq m$	LBLE	18 2F	$Z + (N \oplus V) = 1$	$r > m$	LBGT	18 2E	Signed
$r < m$	LBLT	18 2D	$N \oplus V = 1$	$r \geq m$	LBGE	18 2C	Signed
$r > m$	LBHI	18 22	$C + Z = 0$	$r \leq m$	LBLS	18 23	Unsigned
$r \geq m$	LBHS/LBCC	18 24	$C = 0$	$r < m$	LBLO/LBCS	18 25	Unsigned
$r = m$	LBEQ	18 27	$Z = 1$	$r \neq m$	LBNE	18 26	Unsigned
$r \leq m$	LBLS	18 23	$C + Z = 1$	$r > m$	LBHI	18 22	Unsigned
$r < m$	LBLO/LBCS	18 25	$C = 1$	$r \geq m$	LBHS/LBCC	18 24	Unsigned
Carry	LBCC	18 25	$C = 1$	No Carry	LBCC	18 24	Simple
Negative	LBMI	18 2B	$N = 1$	Plus	LBPL	18 2A	Simple
Overflow	LBVS	18 29	$V = 1$	No Overflow	LBVC	18 28	Simple
$r = 0$	LBEQ	18 27	$Z = 1$	$r \neq 0$	LBNE	18 26	Simple
Always	LBRA	18 20	—	Never	LBRN	18 21	Unconditional

# LBVS

## Long Branch if Overflow Set

# LBVS

**Operation:** If  $V = 1$ , then  $(PC) + \$0004 + Rel \Rightarrow PC$

Simple branch

**Description:** Tests the V status bit and branches if  $V = 1$ .

LBVS causes a branch when a previous operation on two's complement binary values causes an overflow. That is, when LBVS follows a two's complement operation, a branch occurs when the result of the operation is invalid.

See 3.7 **Relative Addressing Mode** for details of branch execution.

### Condition Codes and Boolean Formulas:

S X H I N Z V C

-	-	-	-	-	-	-	-
---	---	---	---	---	---	---	---

None affected.

### Addressing Modes, Machine Code, and Execution Times:

Source Form	Address Mode	Object Code	Cycles	Access Detail
LBVS <i>rel16</i>	REL	18 29 qq rr	4/3	OPPP/OPO <sup>1</sup>

Notes:

1. OPPP/OPO indicates this instruction takes four cycles to refill the instruction queue if the branch is taken and three cycles if the branch is not taken.

Branch				Complementary Branch			
Test	Mnemonic	Opcode	Boolean	Test	Mnemonic	Opcode	Comment
$r > m$	LBGT	18 2E	$Z + (N \oplus V) = 0$	$r \leq m$	LBLE	18 2F	Signed
$r \geq m$	LBGE	18 2C	$N \oplus V = 0$	$r < m$	LBLT	18 2D	Signed
$r = m$	LBEQ	18 27	$Z = 1$	$r \neq m$	LBNE	18 26	Signed
$r \leq m$	LBLE	18 2F	$Z + (N \oplus V) = 1$	$r > m$	LBGT	18 2E	Signed
$r < m$	LBLT	18 2D	$N \oplus V = 1$	$r \geq m$	LBGE	18 2C	Signed
$r > m$	LBHI	18 22	$C + Z = 0$	$r \leq m$	LBLS	18 23	Unsigned
$r \geq m$	LBHS/LBCC	18 24	$C = 0$	$r < m$	LBLO/LBCS	18 25	Unsigned
$r = m$	LBEQ	18 27	$Z = 1$	$r \neq m$	LBNE	18 26	Unsigned
$r \leq m$	LBLS	18 23	$C + Z = 1$	$r > m$	LBHI	18 22	Unsigned
$r < m$	LBLO/LBCS	18 25	$C = 1$	$r \geq m$	LBHS/LBCC	18 24	Unsigned
Carry	LBCS	18 25	$C = 1$	No Carry	LBCC	18 24	Simple
Negative	LBMI	18 2B	$N = 1$	Plus	LBPL	18 2A	Simple
Overflow	LBVS	18 29	$V = 1$	No Overflow	LBVC	18 28	Simple
$r = 0$	LBEQ	18 27	$Z = 1$	$r \neq 0$	LBNE	18 26	Simple
Always	LBRA	18 20	—	Never	LBRN	18 21	Unconditional



# LDAA

Load Accumulator A

# LDAA

**Operation:** (M) ⇒ A

**Description:** Loads the content of memory location M into accumulator A. The condition codes are set according to the data.

**Condition Codes and Boolean Formulas:**

S	X	H	I	N	Z	V	C
—	—	—	—	Δ	Δ	0	—

- N: Set if MSB of result is set; cleared otherwise.  
Z: Set if result is \$00; cleared otherwise.  
V: 0; Cleared.

**Addressing Modes, Machine Code, and Execution Times:**

Source Form	Address Mode	Object Code	Cycles	Access Detail
LDAA #opr8i	IMM	86 ii	1	P
LDAA opr8a	DIR	96 dd	3	rfrP
LDAA opr16a	EXT	B6 hh ll	3	rOP
LDAA oprx0_xysp	IDX	A6 xb	3	rfrP
LDAA oprx9_xysp	IDX1	A6 xb ff	3	rPO
LDAA oprx16_xysp	IDX2	A6 xb ee ff	4	frPP
LDAA [D,xysp]	[D,IDX]	A6 xb	6	fIfrrfP
LDAA [oprx16,xysp]	[IDX2]	A6 xb ee ff	6	fIPrfP

# LDAB

## Load Accumulator B

# LDAB

**Operation:** (M) ⇒ B

**Description:** Loads the content of memory location M into accumulator B. The condition codes are set according to the data.

**Condition Codes and Boolean Formulas:**

S	X	H	I	N	Z	V	C
-	-	-	-	Δ	Δ	0	-

N: Set if MSB of result is set; cleared otherwise.

Z: Set if result is \$00; cleared otherwise.

V: 0; Cleared.

**Addressing Modes, Machine Code, and Execution Times:**

Source Form	Address Mode	Object Code	Cycles	Access Detail
LDAB #opr8i	IMM	C6 ii	1	P
LDAB opr8a	DIR	D6 dd	3	rfP
LDAB opr16a	EXT	F6 hh ll	3	rOP
LDAB oprx0_xysp	IDX	E6 xb	3	rfP
LDAB oprx9,xysp	IDX1	E6 xb ff	3	rPO
LDAB oprx16,xysp	IDX2	E6 xb ee ff	4	frPP
LDAB [D,xysp]	[D,IDX]	E6 xb	6	fIfrfP
LDAB [oprx16,xysp]	[IDX2]	E6 xb ee ff	6	fIPrfP

# LDD

## Load Double Accumulator

# LDD

**Operation:** (M : M + 1) ⇒ A : B

**Description:** Loads the contents of memory locations M and M+1 into double accumulator D. The condition codes are set according to the data. The information from M is loaded into accumulator A, and the information from M+1 is loaded into accumulator B.

### Condition Codes and Boolean Formulas:

S	X	H	I	N	Z	V	C
—	—	—	—	Δ	Δ	0	—

N: Set if MSB of result is set; cleared otherwise.

Z: Set if result is \$0000; cleared otherwise.

V: 0; Cleared.

### Addressing Modes, Machine Code, and Execution Times:

Source Form	Address Mode	Object Code	Cycles	Access Detail
LDD #opr16i	IMM	CC jj kk	2	OP
LDD opr8a	DIR	DC dd	3	RfP
LDD opr16a	EXT	FC hh ll	3	ROP
LDD oprx0_xysp	IDX	EC xb	3	RfP
LDD oprx9,xysp	IDX1	EC xb ff	3	RPO
LDD oprx16,xysp	IDX2	EC xb ee ff	4	fRPP
LDD [D,xysp]	[D,IDX]	EC xb	6	fIfRfP
LDD [oprx16,xysp]	[IDX2]	EC xb ee ff	6	fIPRfP

# LDS

## Load Stack Pointer

# LDS

**Operation:** (M : M+1) ⇒ SP

**Description:** Loads the most significant byte of the SP with the content of memory location M, and loads the least significant byte of the SP with the content of the next byte of memory at M + 1.

### Condition Codes and Boolean Formulas:

S	X	H	I	N	Z	V	C
-	-	-	-	Δ	Δ	0	-

N: Set if MSB of result is set; cleared otherwise.

Z: Set if result is \$0000; cleared otherwise.

V: 0; Cleared.

### Addressing Modes, Machine Code, and Execution Times:

Source Form	Address Mode	Object Code	Cycles	Access Detail
LDS #opr16i	IMM	CF jj kk	2	OP
LDS opr8a	DIR	DF dd	3	RfP
LDS opr16a	EXT	FF hh ll	3	ROP
LDS oprx0,xysp	IDX	EF xb	3	RfP
LDS oprx9,xysp	IDX1	EF xb ff	3	RPO
LDS oprx16,xysp	IDX2	EF xb ee ff	4	fRPP
LDS [D,xysp]	[D,IDX]	EF xb	6	fIfRfP
LDS [oprx16,xysp]	[IDX2]	EF xb ee ff	6	fIPRfP

# LDX

## Load Index Register X

# LDX

**Operation:** (M : M + 1) ⇒ X

**Description:** Loads the most significant byte of index register X with the content of memory location M, and loads the least significant byte of X with the content of the next byte of memory at M + 1.

**Condition Codes and Boolean Formulas:**

S	X	H	I	N	Z	V	C
—	—	—	—	Δ	Δ	0	—

- N: Set if MSB of result is set; cleared otherwise.  
Z: Set if result is \$0000; cleared otherwise.  
V: 0; Cleared.

**Addressing Modes, Machine Code, and Execution Times:**

Source Form	Address Mode	Object Code	Cycles	Access Detail
LDX #opr16i	IMM	CE jj kk	2	OP
LDX opr8a	DIR	DE dd	3	RfP
LDX opr16a	EXT	FE hh ll	3	ROP
LDX oprx0,xysp	IDX	EE xb	3	RfP
LDX oprx9,xysp	IDX1	EE xb ff	3	RPO
LDX oprx16,xysp	IDX2	EE xb ee ff	4	fRPP
LDX [D,xysp]	[D,IDX]	EE xb	6	fIfRfP
LDX [oprx16,xysp]	[IDX2]	EE xb ee ff	6	fIPRfP

# LDY

## Load Index Register Y

# LDY

**Operation:** (M : M + 1) ⇒ Y

**Description:** Loads the most significant byte of index register Y with the content of memory location M, and loads the least significant byte of Y with the content of the next memory location at M + 1.

**Condition Codes and Boolean Formulas:**

S	X	H	I	N	Z	V	C
-	-	-	-	Δ	Δ	0	-

- N: Set if MSB of result is set; cleared otherwise.  
Z: Set if result is \$0000; cleared otherwise.  
V: 0; Cleared.

**Addressing Modes, Machine Code, and Execution Times:**

Source Form	Address Mode	Object Code	Cycles	Access Detail
LDY #opr16i	IMM	CD jj kk	2	OP
LDY opr8a	DIR	DD dd	3	RfP
LDY opr16a	EXT	FD hh ll	3	ROP
LDY oprx0_xysp	IDX	ED xb	3	RfP
LDY oprx9_xysp	IDX1	ED xb ff	3	RPO
LDY oprx16_xysp	IDX2	ED xb ee ff	4	fRPP
LDY [D,xysp]	[D,IDX]	ED xb	6	fIfRfP
LDY [opr16,xysp]	[IDX2]	ED xb ee ff	6	fIPRfP

Source Form	Address Mode	Object Code	Cycles	Access Detail
LDY #opr16i	IMM	CD jj kk	2	OP
LDY opr8a	DIR	DD dd	3	RfP
LDY opr16a	EXT	FD hh ll	3	ROP
LDY oprx0_xysp	IDX	ED xb	3	RfP

# LEAS

Load Stack Pointer with  
Effective Address

# LEAS

**Operation:** Effective Address  $\Rightarrow$  SP

**Description:** Loads the stack pointer with an effective address specified by the program. The effective address can be any indexed addressing mode operand and address except an indirect address. Indexed addressing mode operand addresses are formed by adding an optional constant supplied by the program or an accumulator value to the current value in X, Y, SP, or PC. See **3.8 Indexed Addressing Modes** for more details.

LEAS does not alter condition code bits. This allows stack modification without disturbing CCR bits changed by recent arithmetic operations.

Operation is a bit more complex when LEAS is used with auto-increment or aut-odecrement operand specifications and the SP is the referenced index register. The index register is loaded with what would have gone out to the address bus in the case of a load index instruction. In the case of a pre-increment or pre-decrement, the modification is made before the index register is loaded. In the case of a post-increment or post-decrement, modification would have taken effect after the address went out on the address bus, so post-modification does not affect the content of the index register.

**Condition Codes and Boolean Formulas:**

S	X	H	I	N	Z	V	C
—	—	—	—	—	—	—	—

None affected.

**Addressing Modes, Machine Code, and Execution Times:**

Source Form	Address Mode	Object Code	Cycles	Access Detail
LEAS <i>opr</i> <i>x0</i> _ <i>xy</i> <i>sp</i>	IDX	1B <i>xb</i>	2	PP <sup>1</sup>
LEAS <i>opr</i> <i>x9</i> _ <i>xy</i> <i>sp</i>	IDX1	1B <i>xb ff</i>	2	PO
LEAS <i>opr</i> <i>x16</i> _ <i>xy</i> <i>sp</i>	IDX2	1B <i>xb ee ff</i>	2	PP

Notes:

1. Due to internal CPU requirements, the program word fetch is performed twice to the same address during this instruction.

# LEAX

Load X with Effective Address

# LEAX

**Operation:** Effective Address  $\Rightarrow$  X

**Description:** Loads index register X with an effective address specified by the program. The effective address can be any indexed addressing mode operand and address except an indirect address. Indexed addressing mode operand addresses are formed by adding an optional constant supplied by the program or an accumulator value to the current value in X, Y, SP, or PC. See **3.8 Indexed Addressing Modes** for more details.

Operation is a bit more complex when LEAX is used with auto-increment or auto-decrement operand specifications and index register X is the referenced index register. The index register is loaded with what would have gone out to the address bus in the case of a load indexed instruction. In the case of a pre-increment or pre-decrement, the modification is made before the index register is loaded. In the case of a post-increment or post-decrement, modification would have taken effect after the address went out on the address bus, so post-modification does not affect the content of the index register.

**Condition Codes and Boolean Formulas:**

S	X	H	I	N	Z	V	C
-	-	-	-	-	-	-	-

None affected.

**Addressing Modes, Machine Code, and Execution Times:**

Source Form	Address Mode	Object Code	Cycles	Access Detail
LEAX <i>opr<sub>x</sub>0</i> , <i>xy</i> <i>sp</i>	IDX	1A <i>xb</i>	2	PP <sup>1</sup>
LEAX <i>opr<sub>x</sub>9</i> , <i>xy</i> <i>sp</i>	IDX1	1A <i>xb ff</i>	2	PO
LEAX <i>opr<sub>x</sub>16</i> , <i>xy</i> <i>sp</i>	IDX2	1A <i>xb ee ff</i>	2	PP

Notes:

1. Due to internal CPU requirements, the program word fetch is performed twice to the same address during this instruction.



LEAY

Load Y with Effective Address

LEAY

**Operation:** Effective Address  $\Rightarrow$  Y

**Description:** Loads index register Y with an effective address specified by the program. The effective address can be any indexed addressing mode operand address except an indirect address. Indexed addressing mode operand addresses are formed by adding an optional constant supplied by the program or an accumulator value to the current value in X, Y, SP, or PC. See **3.8 Indexed Addressing Modes** for more details.

Operation is a bit more complex when LEAY is used with auto-increment or auto-decrement operand specifications and index register Y is the referenced index register. The index register is loaded with what would have gone out to the address bus in the case of a load indexed instruction. In the case of a pre-increment or pre-decrement, the modification is made before the index register is loaded. In the case of a post-increment or post-decrement, modification would have taken effect after the address went out on the address bus, so post-modification does not affect the content of the index register.

**Condition Codes and Boolean Formulas:**

S	X	H	I	N	Z	V	C
—	—	—	—	—	—	—	—

None affected.

**Addressing Modes, Machine Code, and Execution Times:**

Source Form	Address Mode	Object Code	Cycles	Access Detail
LEAY <i>oprx0,xysp</i>	IDX	19 xb	2	PP <sup>1</sup>
LEAY <i>oprx9,xysp</i>	IDX1	19 xb ff	2	PO
LEAY <i>oprx16,xysp</i>	IDX2	19 xb ee ff	2	PP

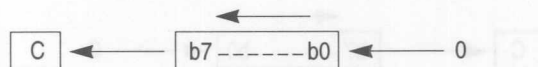
- Notes:
1. Due to internal CPU requirements, the program word fetch is performed twice to the same address during this instruction.

# LSL

## Logical Shift Left Memory (Same as ASL)

# LSL

### Operation:



**Description:** Shifts all bits of the memory location M one place to the left. Bit 0 is loaded with zero. The C status bit is loaded from the most significant bit of M.

### Condition Codes and Boolean Formulas:

S	X	H	I	N	Z	V	C
-	-	-	-	$\Delta$	$\Delta$	$\Delta$	$\Delta$

**N:** Set if MSB of result is set; cleared otherwise.

**Z:** Set if result is \$00; cleared otherwise.

**V:**  $N \oplus C = [N \bullet \overline{C}] + [\overline{N} \bullet C]$  (for N and C after the shift)

Set if (N is set and C is cleared) or (N is cleared and C is set); cleared otherwise (for values of N and C after the shift).

**C:** M7

Set if the LSB of M was set before the shift; cleared otherwise.

### Addressing Modes, Machine Code, and Execution Times:

Source Form	Address Mode	Object Code	Cycles	Access Detail
LSL <i>opr16a</i>	EXT	78 hh ll	4	rOPw
LSL <i>opr0_xysp</i>	IDX	68 xb	3	rPw
LSL <i>opr9_xysp</i>	IDX1	68 xb ff	4	rPOw
LSL <i>opr16_xysp</i>	IDX2	68 xb ee ff	5	frPPw
LSL [D, <i>xysp</i> ]	[D,IDX]	68 xb	6	fIFrPw
LSL [ <i>opr16_xysp</i> ]	[IDX2]	68 xb ee ff	6	fIPrPw

# LSLA

Logical Shift Left A  
(Same as ASLA)

# LSLA

**Operation:**



**Description:** Shifts all bits of accumulator A one place to the left. Bit 0 is loaded with zero. The C status bit is loaded from the most significant bit of A.

**Condition Codes and Boolean Formulas:**

S	X	H	I	N	Z	V	C
-	-	-	-	Δ	Δ	Δ	Δ

- N: Set if MSB of result is set; cleared otherwise.
- Z: Set if result is \$00; cleared otherwise.
- V:  $N \oplus C = [N \bullet \overline{C}] + [\overline{N} \bullet C]$  (for N and C after the shift)  
Set if (N is set and C is cleared) or (N is cleared and C is set); cleared otherwise (for values of N and C after the shift).
- C: A7  
Set if the LSB of A was set before the shift; cleared otherwise.

**Addressing Modes, Machine Code, and Execution Times:**

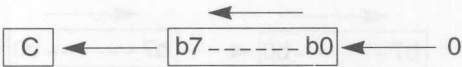
Source Form	Address Mode	Object Code	Cycles	Access Detail
LSLA	INH	48	1	0

# LSLB

Logical Shift Left B  
(Same as ASLB)

# LSLB

Operation:



**Description:** Shifts all bits of accumulator B one place to the left. Bit 0 is loaded with zero. The C status bit is loaded from the most significant bit of B.

Condition Codes and Boolean Formulas:

S	X	H	I	N	Z	V	C
-	-	-	-	$\Delta$	$\Delta$	$\Delta$	$\Delta$

- N: Set if MSB of result is set; cleared otherwise.
- Z: Set if result is \$00; cleared otherwise.
- V:  $N \oplus C = [N \cdot \overline{C}] + [\overline{N} \cdot C]$  (for N and C after the shift)  
Set if (N is set and C is cleared) or (N is cleared and C is set); cleared otherwise (for values of N and C after the shift).
- C: B7  
Set if the LSB of B was set before the shift; cleared otherwise.

Addressing Modes, Machine Code, and Execution Times:

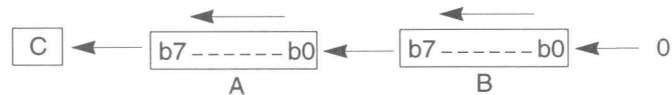
Source Form	Address Mode	Object Code	Cycles	Access Detail
LSLB	INH	58	1	0

# LSLD

Logical Shift Left Double  
(Same as ASLD)

# LSLD

**Operation:**



**Description:** Shifts all bits of double accumulator D one place to the left. Bit 0 is loaded with zero. The C status bit is loaded from the most significant bit of accumulator A.

**Condition Codes and Boolean Formulas:**

S	X	H	I	N	Z	V	C
-	-	-	-	$\Delta$	$\Delta$	$\Delta$	$\Delta$

- N: Set if MSB of result is set; cleared otherwise.
- Z: Set if result is \$0000; cleared otherwise.
- V:  $N \oplus C = [N \bullet \overline{C}] + [\overline{N} \bullet C]$  (for N and C after the shift)  
Set if (N is set and C is cleared) or (N is cleared and C is set); cleared otherwise (for values of N and C after the shift).
- C: D15  
Set if the MSB of D was set before the shift; cleared otherwise.

**Addressing Modes, Machine Code, and Execution Times:**

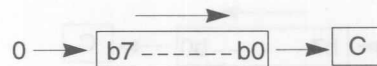
Source Form	Address Mode	Object Code	Cycles	Access Detail
LSLD	INH	59	1	0

# LSR

## Logical Shift Right Memory

# LSR

### Operation:



**Description:** Shifts all bits of memory location M one place to the right. Bit 7 is loaded with zero. The C status bit is loaded from the least significant bit of M.

### Condition Codes and Boolean Formulas:

S	X	H	I	N	Z	V	C
-	-	-	-	0	$\Delta$	$\Delta$	$\Delta$

N: 0; Cleared.

Z: Set if result is \$00; cleared otherwise.

V:  $N \oplus C = [N \bullet \bar{C}] + [\bar{N} \bullet C]$  (for N and C after the shift)

Set if (N is set and C is cleared) or (N is cleared and C is set); cleared otherwise (for values of N and C after the shift).

C: M0

Set if the LSB of M was set before the shift; cleared otherwise.

### Addressing Modes, Machine Code, and Execution Times:

Source Form	Address Mode	Object Code	Cycles	Access Detail
LSR <i>opr16a</i>	EXT	74 hh ll	4	rOPw
LSR <i>opr0_xysp</i>	IDX	64 xb	3	rPw
LSR <i>opr9,xysp</i>	IDX1	64 xb ff	4	rPOw
LSR <i>opr16,xysp</i>	IDX2	64 xb ee ff	5	frPPw
LSR [D,xysp]	[D,IDX]	64 xb	6	fIfPrPw
LSR [ <i>opr16,xysp</i> ]	[IDX2]	64 xb ee ff	6	fIPrPw

# LSRA

Logical Shift Right A

# LSRA

Operation:



**Description:** Shifts all bits of accumulator A one place to the right. Bit 7 is loaded with zero. The C status bit is loaded from the least significant bit of A.

Condition Codes and Boolean Formulas:

S	X	H	I	N	Z	V	C
-	-	-	-	0	$\Delta$	$\Delta$	$\Delta$

- N: 0; Cleared.
- Z: Set if result is \$00; cleared otherwise.
- V:  $N \oplus C = [N \bullet \overline{C}] + [\overline{N} \bullet C]$  (for N and C after the shift)  
Set if (N is set and C is cleared) or (N is cleared and C is set); cleared otherwise (for values of N and C after the shift).
- C: A0  
Set if the LSB of A was set before the shift; cleared otherwise.

Addressing Modes, Machine Code, and Execution Times:

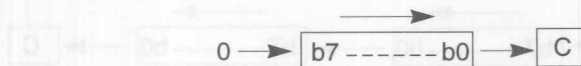
Source Form	Address Mode	Object Code	Cycles	Access Detail
LSRA	INH	44	1	0

# LSRB

Logical Shift Right B

# LSRB

## Operation:



**Description:** Shifts all bits of accumulator B one place to the right. Bit 7 is loaded with zero. The C status bit is loaded from the least significant bit of B.

## Condition Codes and Boolean Formulas:

S	X	H	I	N	Z	V	C
-	-	-	-	0	$\Delta$	$\Delta$	$\Delta$

N: 0; Cleared.

Z: Set if result is \$00; cleared otherwise.

V:  $N \oplus C = [N \cdot \bar{C}] + [\bar{N} \cdot C]$  (for N and C after the shift)  
Set if (N is set and C is cleared) or (N is cleared and C is set); cleared otherwise (for values of N and C after the shift).

C: B0

Set if the LSB of B was set before the shift; cleared otherwise.

## Addressing Modes, Machine Code, and Execution Times:

Source Form	Address Mode	Object Code	Cycles	Access Detail
LSRB	INH	54	1	0

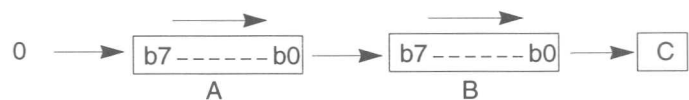


# LSRD

Logical Shift Right Double

# LSRD

Operation:



**Description:** Shifts all bits of double accumulator D one place to the right. D15 (MSB of A) is loaded with zero. The C status bit is loaded from D0 (LSB of B).

Condition Codes and Boolean Formulas:

S	X	H	I	N	Z	V	C
—	—	—	—	0	Δ	Δ	Δ

- N: 0; Cleared.
- Z: Set if result is \$0000; cleared otherwise.
- V: D0  
Set if, after the shift operation, C is set; cleared otherwise.
- C: D0  
Set if the LSB of D was set before the shift; cleared otherwise.

Addressing Modes, Machine Code, and Execution Times:

Source Form	Address Mode	Object Code	Cycles	Access Detail
LSRD	INH	49	1	0

# MAXA

Place Larger of Two  
Unsigned 8-Bit Values  
in Accumulator A

# MAXA

**Operation:** MAX ((A), (M))  $\Rightarrow$  A

**Description:** Subtracts an unsigned 8-bit value in memory from an unsigned 8-bit value in accumulator A to determine which is larger, and leaves the larger of the two values in A. The Z status bit is set when the result of the subtraction is zero (the values are equal), and the C status bit is set when the subtraction requires a borrow (the value in memory is larger than the value in the accumulator). When C = 1, the value in A has been replaced by the value in memory.

The unsigned value in memory is accessed by means of indexed addressing modes, which allow a great deal of flexibility in specifying the address of the operand. Auto increment/decrement variations of indexed addressing facilitate finding the largest value in a list of values.

## Condition Codes and Boolean Formulas:

S	X	H	I	N	Z	V	C
—	—	—	—	$\Delta$	$\Delta$	$\Delta$	$\Delta$

N: Set if MSB of result is set; cleared otherwise.

Z: Set if result is \$00; cleared otherwise.

V:  $X7 \cdot \overline{M7} \cdot \overline{R7} + \overline{X7} \cdot M7 \cdot R7$

Set if a two's complement overflow resulted from the operation; cleared otherwise.

C:  $\overline{X7} \cdot M7 + M7 \cdot R7 + R7 \cdot \overline{X7}$

Set if the value of the content of memory is larger than the value of the accumulator; cleared otherwise.

Condition codes reflect internal subtraction ( $R = A - M$ ).

## Addressing Modes, Machine Code, and Execution Times:

Source Form	Address Mode	Object Code	Cycles	Access Detail
MAXA <i>opr0,xysp</i>	IDX	18 18 xb	4	OrfP
MAXA <i>opr9,xysp</i>	IDX1	18 18 xb ff	4	OrPO
MAXA <i>opr16,xysp</i>	IDX2	18 18 xb ee ff	5	OfrPP
MAXA [D,xysp]	[D,IDX]	18 18 xb	7	OfIfrrfP
MAXA [ <i>opr16,xysp</i> ]	[IDX2]	18 18 xb ee ff	7	OfIPrfP

# MAXM

Place Larger of Two  
Unsigned 8-Bit Values  
in Memory

# MAXM

**Operation:** MAX ((A), (M))  $\Rightarrow$  M

**Description:** Subtracts an unsigned 8-bit value in memory from an unsigned 8-bit value in accumulator A to determine which is larger, and leaves the larger of the two values in the memory location. The Z status bit is set when the result of the subtraction is zero (the values are equal), and the C status bit is set when the subtraction requires a borrow (the value in memory is larger than the value in the accumulator). When C = 0, the value in accumulator A has replaced the value in memory.

The unsigned value in memory is accessed by means of indexed addressing modes, which allow a great deal of flexibility in specifying the address of the operand.

## Condition Codes and Boolean Formulas:

S	X	H	I	N	Z	V	C
—	—	—	—	$\Delta$	$\Delta$	$\Delta$	$\Delta$

N: Set if MSB of result is set; cleared otherwise.

Z: Set if result is \$00; cleared otherwise.

V:  $X7 \bullet \overline{M7} \bullet \overline{R7} + \overline{X7} \bullet M7 \bullet R7$   
Set if a two's complement overflow resulted from the operation;  
cleared otherwise.

C:  $\overline{X7} \bullet M7 + M7 \bullet R7 + R7 \bullet \overline{X7}$   
Set if the value of the content of memory is larger than the value of  
the accumulator; cleared otherwise.

Condition codes reflect internal subtraction ( $R = A - M$ ).

## Addressing Modes, Machine Code, and Execution Times:

Source Form	Address Mode	Object Code	Cycles	Access Detail
MAXM <i>opr</i> x0, <i>xy</i> sp	IDX	18 1C xb	4	OrPw
MAXM <i>opr</i> x9, <i>xy</i> sp	IDX1	18 1C xb ff	5	OrPwO
MAXM <i>opr</i> x16, <i>xy</i> sp	IDX2	18 1C xb ee ff	6	OfxPwP
MAXM [D, <i>xy</i> sp]	[D,IDX]	18 1C xb	7	OfIfxPw
MAXM [ <i>opr</i> x16, <i>xy</i> sp]	[IDX2]	18 1C xb ee ff	7	OfIPwPw

# MEM

## Determine Grade of Membership (Fuzzy Logic)

# MEM

**Operation:** Grade of Membership  $\Rightarrow M(Y)$   
(Y) + \$0001  $\Rightarrow Y$   
(X) + \$0004  $\Rightarrow X$

**Description:** Accumulator A and index registers X and Y must be set up as follows before executing MEM.

A must hold the current crisp value of a system input variable.

X must point to a 4-byte data structure that describes the trapezoidal membership function for a label of the system input.

Y must point to the fuzzy input (RAM location) where the resulting grade of membership is to be stored.

The 4-byte membership function data structure consists of Point\_1, Point\_2, Slope\_1, and Slope\_2, in that order.

Point\_1 is the X-axis starting point for the leading side of the trapezoid, and Slope\_1 is the slope of the leading side of the trapezoid.

Point\_2 is the X-axis position of the rightmost point of the trapezoid, and Slope\_2 is the slope of the trailing side of the trapezoid. The trailing side slopes up and left from Point\_2.

A Slope\_1 or Slope\_2 value of \$00 indicates a special case where the membership function either starts with a grade of \$FF at input = Point\_1, or ends with a grade of \$FF at input = Point\_2 (infinite slope).

When MEM is executed, X points at Point\_1 and Slope\_2 is at X + 3. After execution, the content of A is unchanged. X has been incremented by four to point to the next set of membership function points and slopes. The fuzzy input (RAM location) to which Y pointed contains the grade of membership that was calculated by MEM, and Y has been incremented by one so it points to the next fuzzy input.

### Condition Codes and Boolean Formulas:

S	X	H	I	N	Z	V	C
-	-	?	-	?	?	?	?

H, N, Z, V, and C may be altered by this instruction.

### Addressing Modes, Machine Code, and Execution Times:

Source Form	Address Mode	Object Code	Cycles	Access Detail
MEM	Special	01	5	RRFOW

# MINA

## Place Smaller of Two Unsigned 8-Bit Values in Accumulator A

# MINA

**Operation:** MIN ((A), (M)) ⇒ A

**Description:** Subtracts an unsigned 8-bit value in memory from an unsigned 8-bit value in accumulator A to determine which is larger, and leaves the smaller of the two values in accumulator A. The Z status bit is set when the result of the subtraction is zero (the values are equal), and the C status bit is set when the subtraction requires a borrow (the value in memory is larger than the value in the accumulator). When C = 0, the value in accumulator A has been replaced by the value in memory.

The unsigned value in memory is accessed by means of indexed addressing modes, which allow a great deal of flexibility in specifying the address of the operand. Auto increment/decrement variations of indexed addressing facilitate finding the largest value in a list of values.

### Condition Codes and Boolean Formulas:

S	X	H	I	N	Z	V	C
—	—	—	—	Δ	Δ	Δ	Δ

- N: Set if MSB of result is set; cleared otherwise.
- Z: Set if result is \$00; cleared otherwise.
- V:  $X7 \bullet \overline{M7} \bullet \overline{R7} + \overline{X7} \bullet M7 \bullet R7$   
Set if a two's complement overflow resulted from the operation; cleared otherwise.
- C:  $\overline{X7} \bullet M7 + M7 \bullet R7 + R7 \bullet \overline{X7}$   
Set if the value of the content of memory is larger than the value of the accumulator; cleared otherwise.

Condition codes reflect internal subtraction (R = A – M).

### Addressing Modes, Machine Code, and Execution Times:

Source Form	Address Mode	Object Code	Cycles	Access Detail
MINA <i>opr</i> x0_ <i>xy</i> sp	IDX	18 19 xb	4	OrfP
MINA <i>opr</i> x9_ <i>xy</i> sp	IDX1	18 19 xb ff	4	OrPO
MINA <i>opr</i> x16_ <i>xy</i> sp	IDX2	18 19 xb ee ff	5	OfrPP
MINA [D_ <i>xy</i> sp]	[D,IDX]	18 19 xb	7	OfIfrrfP
MINA [ <i>opr</i> x16_ <i>xy</i> sp]	[IDX2]	18 19 xb ee ff	7	OfIPrfP

# MINM

Place Smaller of Two  
Unsigned 8-Bit Values  
in Memory

# MINM

**Operation:**  $\text{MIN} ((A), (M)) \Rightarrow M$

**Description:** Subtracts an unsigned 8-bit value in memory from an unsigned 8-bit value in accumulator A to determine which is larger, and leaves the smaller of the two values in the memory location. The Z status bit is set when the result of the subtraction is zero (the values are equal), and the C status bit is set when the subtraction requires a borrow (the value in memory is larger than the value in the accumulator). When C = 1, the value in accumulator A has replaced the value in memory.

The unsigned value in memory is accessed by means of indexed addressing modes, which allow a great deal of flexibility in specifying the address of the operand.

## Condition Codes and Boolean Formulas:

S	X	H	I	N	Z	V	C
—	—	—	—	$\Delta$	$\Delta$	$\Delta$	$\Delta$

N: Set if MSB of result is set; cleared otherwise.

Z: Set if result is \$00; cleared otherwise.

V:  $X7 \cdot \overline{M7} \cdot R7 + \overline{X7} \cdot M7 \cdot R7$   
Set if a two's complement overflow resulted from the operation; cleared otherwise.

C:  $\overline{X7} \cdot M7 + M7 \cdot R7 + R7 \cdot \overline{X7}$   
Set if the value of the content of memory is larger than the value of the accumulator; cleared otherwise.

Condition codes reflect internal subtraction ( $R = A - M$ ).

## Addressing Modes, Machine Code, and Execution Times:

Source Form	Address Mode	Object Code	Cycles	Access Detail
MINM <i>opr</i> x0, <i>xy</i> sp	IDX	18 1D xb	4	OrPw
MINM <i>opr</i> x9, <i>xy</i> sp	IDX1	18 1D xb ff	5	OrPwO
MINM <i>opr</i> x16, <i>xy</i> sp	IDX2	18 1D xb ee ff	6	OfPrPwP
MINM [D, <i>xy</i> sp]	[D,IDX]	18 1D xb	7	OfIfrPw
MINM [ <i>opr</i> x16, <i>xy</i> sp]	[IDX2]	18 1D xb ee ff	7	OfIPrPw

MOVB

Move a Byte of Data  
from One Memory Location to Another

MOVB

**Operation:** (M<sub>1</sub>) ⇒ M<sub>2</sub>

**Description:** Moves the content of one memory location to another memory location. The content of the source memory location is not changed.

Move instructions use separate addressing modes to access the source and destination of a move. The following combinations of addressing modes are supported: IMM-EXT, IMM-IDX, EXT-EXT, EXT-IDX, IDX-EXT, and IDX-IDX. IDX operands allow indexed addressing mode specifications that fit in a single postbyte; including 5-bit constant, accumulator offsets, and auto increment/decrement modes. Nine-bit and 16-bit constant offsets would require additional extension bytes and are not allowed. Indexed indirect modes (for example [D,r]) are also not allowed.

There are special considerations when using PC-relative addressing with move instructions. These are discussed in **3.9 Instructions Using Multiple Modes**.

**Condition Codes and Boolean Formulas:**

S	X	H	I	N	Z	V	C
-	-	-	-	-	-	-	-

None affected.

**Addressing Modes, Machine Code, and Execution Times:**

Source Form <sup>1</sup>	Address Mode	Object Code	Cycles	Access Detail
MOVB #opr8, opr16a	IMM-EXT	18 0B ii hh ll	4	OPwP
MOVB #opr8i, oprx0_xysp	IMM-IDX	18 08 xb ii	4	OPwO
MOVB opr16a, opr16a	EXT-EXT	18 0C hh ll hh ll	6	OrPwPO
MOVB opr16a, oprx0_xysp	EXT-IDX	18 09 xb hh ll	5	OPrPw
MOVB oprx0_xysp, opr16a	IDX-EXT	18 0D xb hh ll	5	OrPwP
MOVB oprx0_xysp, oprx0_xysp	IDX-IDX	18 0A xb xb	5	OrPwO

Notes:  
1. The first operand in the source code statement specifies the source for the move.

# MOVW Move a Word of Data from One Memory Location to Another MOVW

**Operation:**  $(M : M + 1_1) \Rightarrow M : M + 1_2$

**Description:** Moves the content of one location in memory to another location in memory. The content of the source memory location is not changed.

Move instructions use separate addressing modes to access the source and destination of a move. The following combinations of addressing modes are supported: IMM-EXT, IMM-IDX, EXT-EXT, EXT-IDX, IDX-EXT, and IDX-IDX. IDX operands allow indexed addressing mode specifications that fit in a single postbyte; including 5-bit constant, accumulator offsets, and auto increment/decrement modes. Nine-bit and 16-bit constant offsets would require additional extension bytes and are not allowed. Indexed indirect modes (for example [D,r]) are also not allowed.

There are special considerations when using PC-relative addressing with move instructions. These are discussed in **3.9 Instructions Using Multiple Modes**.

## Condition Codes and Boolean Formulas:

S	X	H	I	N	Z	V	C
-	-	-	-	-	-	-	-

None affected.

## Addressing Modes, Machine Code, and Execution Times:

Source Form <sup>1</sup>	Address Mode	Object Code	Cycles	Access Detail
MOVW #opr16i, opr16a	IMM-EXT	18 03 jj kk hh ll	5	OPWPO
MOVW #opr16i, oprx0_ysp	IMM-IDX	18 00 xb jj kk	4	OPPW
MOVW opr16a, opr16a	EXT-EXT	18 04 hh ll hh ll	6	ORPWPO
MOVW opr16a, oprx0_ysp	EXT-IDX	18 01 xb hh ll	5	OPRPW
MOVW oprx0_ysp, opr16a	IDX-EXT	18 05 xb hh ll	5	ORPWP
MOVW oprx0_ysp, oprx0_ysp	IDX-IDX	18 02 xb xb	5	ORPWO

Notes:

1. The first operand in the source code statement specifies the source for the move.



# MUL

Multiply  
8-Bit by 8-Bit (Unsigned)

# MUL

**Operation:**  $(A) \times (B) \Rightarrow A : B$

**Description:** Multiplies the 8-bit unsigned binary value in accumulator A by the 8-bit unsigned binary value in accumulator B, and places the 16-bit unsigned result in double accumulator D. The carry flag allows rounding the most significant byte of the result through the sequence: MUL, ADCA #0.

## Condition Codes and Boolean Formulas:

S	X	H	I	N	Z	V	C
—	—	—	—	—	—	—	$\Delta$

C: R7  
Set if bit 7 of the result (B bit 7) is set; cleared otherwise.

## Addressing Modes, Machine Code, and Execution Times:

Source Form	Address Mode	Object Code	Cycles	Access Detail
MUL	INH	12	3	££0

# NEG

## Negate Memory

# NEG

**Operation:**  $0 - (M) = (\overline{M}) + 1 \Rightarrow M$

**Description:** Replaces the content of memory location M with its two's complement (the value \$80 is left unchanged).

### Condition Codes and Boolean Formulas:

S	X	H	I	N	Z	V	C
-	-	-	-	$\Delta$	$\Delta$	$\Delta$	$\Delta$

N: Set if MSB of result is set; cleared otherwise.

Z: Set if result is \$00; cleared otherwise.

V:  $R7 \bullet R6 \bullet R5 \bullet R4 \bullet R3 \bullet R2 \bullet R1 \bullet R0$

Set if there is a two's complement overflow from the implied subtraction from zero; cleared otherwise. Two's complement overflow occurs if and only if  $(M) = \$80$

C:  $R7 + R6 + R5 + R4 + R3 + R2 + R1 + R0$

Set if there is a borrow in the implied subtraction from zero; cleared otherwise. Set in all cases except when  $(M) = \$00$ .

### Addressing Modes, Machine Code, and Execution Times:

Source Form	Address Mode	Object Code	Cycles	Access Detail
NEG <i>opr16a</i>	EXT	70 hh ll	4	rOPw
NEG <i>opr0_xysp</i>	IDX	60 xb	3	rPw
NEG <i>opr9,xysp</i>	IDX1	60 xb ff	4	rPOw
NEG <i>opr16,xysp</i>	IDX2	60 xb ee ff	5	frPPw
NEG [D, <i>xysp</i> ]	[D,IDX]	60 xb	6	fIfPrPw
NEG [ <i>opr16,xysp</i> ]	[IDX2]	60 xb ee ff	6	fIPrPw

# NEGA

Negate A

# NEGA

**Operation:**  $0 - (A) = (\overline{A}) + 1 \Rightarrow A$

**Description:** Replaces the content of accumulator A with its two's complement (the value \$80 is left unchanged).

**Condition Codes and Boolean Formulas:**

S	X	H	I	N	Z	V	C
-	-	-	-	Δ	Δ	Δ	Δ

- N: Set if MSB of result is set; cleared otherwise.
- Z: Set if result is \$00; cleared otherwise.
- V:  $R7 \bullet R6 \bullet R5 \bullet R4 \bullet R3 \bullet R2 \bullet R1 \bullet R0$   
Set if there is a two's complement overflow from the implied subtraction from zero; cleared otherwise. Two's complement overflow occurs if and only if (A) = \$80.
- C:  $R7 + R6 + R5 + R4 + R3 + R2 + R1 + R0$   
Set if there is a borrow in the implied subtraction from zero; cleared otherwise. Set in all cases except when (A) = \$00.

**Addressing Modes, Machine Code, and Execution Times:**

Source Form	Address Mode	Object Code	Cycles	Access Detail
NEGA	INH	40	1	0

# NEGB

Negate B

# NEGB

**Operation:**  $0 - (B) = (\overline{B}) + 1 \Rightarrow B$

**Description:** Replaces the content of accumulator B with its two's complement (the value \$80 is left unchanged).

**Condition Codes and Boolean Formulas:**

S	X	H	I	N	Z	V	C
-	-	-	-	Δ	Δ	Δ	Δ

N: Set if MSB of result is set; cleared otherwise.

Z: Set if result is \$00; cleared otherwise.

V:  $R7 \bullet R6 \bullet R5 \bullet R4 \bullet R3 \bullet R2 \bullet R1 \bullet R0$

Set if there is a two's complement overflow from the implied subtraction from zero; cleared otherwise. Two's complement overflow occurs if and only if  $(B) = \$80$ .

C:  $R7 + R6 + R5 + R4 + R3 + R2 + R1 + R0$

Set if there is a borrow in the implied subtraction from zero; cleared otherwise. Set in all cases except when  $(B) = \$00$ .

**Addressing Modes, Machine Code, and Execution Times:**

Source Form	Address Mode	Object Code	Cycles	Access Detail
NEGB	INH	50	1	0

# NOP

Null Operation

# NOP

**Operation:** No operation

**Description:** This single-byte instruction increments the PC and does nothing else. No other CPU registers are affected. NOP is typically used to produce a time delay, although some software disciplines discourage CPU frequency-based time delays. During debug, NOP instructions are sometimes used to temporarily replace other machine code instructions, thus disabling the replaced instruction(s).

**Condition Codes and Boolean Formulas:**

S	X	H	I	N	Z	V	C
—	—	—	—	—	—	—	—

None affected.

**Addressing Modes, Machine Code, and Execution Times:**

Source Form	Address Mode	Object Code	Cycles	Access Detail
NOP	INH	A7	1	0

# ORAA

Inclusive OR A

# ORAA

**Operation:**  $(A) + (M) \Rightarrow A$

**Description:** Performs bitwise logical inclusive OR between the content of accumulator A and the content of memory location M and places the result in A. Each bit of A after the operation is the logical inclusive OR of the corresponding bits of M and of A before the operation.

**Condition Codes and Boolean Formulas:**

S	X	H	I	N	Z	V	C
-	-	-	-	$\Delta$	$\Delta$	0	-

N: Set if MSB of result is set; cleared otherwise.

Z: Set if result is \$00; cleared otherwise.

V: 0; Cleared.

**Addressing Modes, Machine Code, and Execution Times:**

Source Form	Address Mode	Object Code	Cycles	Access Detail
ORAA #opr8i	IMM	8A ii	1	P
ORAA opr8a	DIR	9A dd	3	rfrP
ORAA opr16a	EXT	BA hh ll	3	rOP
ORAA oprx0_xysp	IDX	AA xb	3	rfrP
ORAA oprx9_xysp	IDX1	AA xb ff	3	rPO
ORAA oprx16_xysp	IDX2	AA xb ee ff	4	frPP
ORAA [D,xysp]	[D,IDX]	AA xb	6	fIfrrP
ORAA [oprx16,xysp]	[IDX2]	AA xb ee ff	6	fIPrrP

# ORAB

Inclusive OR B

# ORAB

**Operation:** (B) + (M)  $\Rightarrow$  B

**Description:** Performs bitwise logical inclusive OR between the content of accumulator B and the content of memory location M. The result is placed in B. Each bit of B after the operation is the logical inclusive OR of the corresponding bits of M and of B before the operation.

**Condition Codes and Boolean Formulas:**

S	X	H	I	N	Z	V	C
—	—	—	—	$\Delta$	$\Delta$	0	—

N: Set if MSB of result is set; cleared otherwise.

Z: Set if result is \$00; cleared otherwise.

V: 0; Cleared.

**Addressing Modes, Machine Code, and Execution Times:**

Source Form	Address Mode	Object Code	Cycles	Access Detail
ORAB #opr8i	IMM	CA ii	1	P
ORAB opr8a	DIR	DA dd	3	rfrP
ORAB opr16a	EXT	FA hh ll	3	rOP
ORAB oprx0_xysp	IDX	EA xb	3	rfrP
ORAB oprx9_xysp	IDX1	EA xb ff	3	rPO
ORAB oprx16_xysp	IDX2	EA xb ee ff	4	frPP
ORAB [D,xysp]	[D,IDX]	EA xb	6	fIfrrfP
ORAB [oprx16,xysp]	[IDX2]	EA xb ee ff	6	fIPrfP

# ORCC

## Logical OR CCR with Mask

# ORCC

**Operation:** (CCR) + (M) ⇒ CCR

**Description:** Performs bitwise logical inclusive OR between the content of memory location M and the content of the CCR, and places the result in the CCR. Each bit of the CCR after the operation is the logical OR of the corresponding bits of M and of CCR before the operation. To set one or more bits, set the corresponding bit of the mask equal to one. Bits corresponding to zeros in the mask are not changed by the ORCC operation.

**Condition Codes and Boolean Formulas:**

S	X	H	I	N	Z	V	C
↑	—	↑	↑	↑	↑	↑	↑

Condition code bits are set if the corresponding bit was one before the operation or if the corresponding bit in the instruction-provided mask is one. The X interrupt mask cannot be set by any software instruction.

**Addressing Modes, Machine Code, and Execution Times:**

Source Form	Address Mode	Object Code	Cycles	Access Detail
ORCC #opr8i	IMM	14 ii	1	P



# PSHA

Push A onto Stack

# PSHA

**Operation:** (SP) – \$0001 ⇒ SP  
(A) ⇒ M(SP)

**Description:** Stacks the content of accumulator A. The stack pointer is decremented by one. The content of A is then stacked at the address the SP points to.

Push instructions are commonly used to save the contents of one or more CPU registers at the start of a subroutine. Complementary pull instructions can be used to restore the saved CPU registers just before returning from the subroutine.

**Condition Codes and Boolean Formulas:**

S	X	H	I	N	Z	V	C
–	–	–	–	–	–	–	–

None affected.

**Addressing Modes, Machine Code, and Execution Times:**

Source Form	Address Mode	Object Code	Cycles	Access Detail
PSHA	INH	36	2	0s

# PSHB

Push B onto Stack

# PSHB

**Operation:**  $(SP) - \$0001 \Rightarrow SP$   
 $(B) \Rightarrow M_{(SP)}$

**Description:** Stacks the content of accumulator B. The stack pointer is decremented by one. The content of B is then stacked at the address the SP points to.

Push instructions are commonly used to save the contents of one or more CPU registers at the start of a subroutine. Complementary pull instructions can be used to restore the saved CPU registers just before returning from the subroutine.

**Condition Codes and Boolean Formulas:**

S	X	H	I	N	Z	V	C
-	-	-	-	-	-	-	-

None affected.

**Addressing Modes, Machine Code, and Execution Times:**

Source Form	Address Mode	Object Code	Cycles	Access Detail
PSHB	INH	37	2	0s

# PSHC

Push CCR onto Stack

# PSHC

**Operation:**  $(SP) - \$0001 \Rightarrow SP$   
 $(CCR) \Rightarrow M_{(SP)}$

**Description:** Stacks the content of the condition codes register. The stack pointer is decremented by one. The content of the CCR is then stacked at the address to which the SP points.

Push instructions are commonly used to save the contents of one or more CPU registers at the start of a subroutine. Complementary pull instructions can be used to restore the saved CPU registers just before returning from the subroutine.

## Condition Codes and Boolean Formulas:

S	X	H	I	N	Z	V	C
—	—	—	—	—	—	—	—

None affected.

## Addressing Modes, Machine Code, and Execution Times:

Source Form	Address Mode	Object Code	Cycles	Access Detail
PSHC	INH	39	2	0s

# PSHD

Push Double Accumulator onto Stack

# PSHD

**Operation:** (SP) - \$0002  $\Rightarrow$  SP  
(A : B)  $\Rightarrow$  M(SP) : M(SP + 1)

**Description:** Stacks the content of double accumulator D. The stack pointer is decremented by two, then the contents of accumulators A and B are stacked at the location to which the SP points.

After PSHD executes, the SP points to the stacked value of accumulator A. This stacking order is the opposite of the order in which A and B are stacked when an interrupt is recognized. The interrupt stacking order is backward-compatible with the M6800, which had no 16-bit accumulator.

Push instructions are commonly used to save the contents of one or more CPU registers at the start of a subroutine. Complementary pull instructions can be used to restore the saved CPU registers just before returning from the subroutine.

**Condition Codes and Boolean Formulas:**

S	X	H	I	N	Z	V	C
-	-	-	-	-	-	-	-

None affected.		
----------------	--	--

**Addressing Modes, Machine Code, and Execution Times:**

Source Form	Address Mode	Object Code	Cycles	Access Detail
PSHD	INH	3B	2	OS

PSHX

Push Index Register X onto Stack

PSHX

**Operation:**  $(SP) - \$0002 \Rightarrow SP$   
 $(X_H : X_L) \Rightarrow M_{(SP)} : M_{(SP + 1)}$

**Description:** Stacks the content of index register X. The stack pointer is decremented by two. The content of X is then stacked at the address to which the SP points. After PSHX executes, the SP points to the stacked value of the high-order half of X.

Push instructions are commonly used to save the contents of one or more CPU registers at the start of a subroutine. Complementary pull instructions can be used to restore the saved CPU registers just before returning from the subroutine.

**Condition Codes and Boolean Formulas:**

S	X	H	I	N	Z	V	C
—	—	—	—	—	—	—	—

None affected.

**Addressing Modes, Machine Code, and Execution Times:**

Source Form	Address Mode	Object Code	Cycles	Access Detail
PSHX	INH	34	2	OS

# PSHY

Push Index Register Y onto Stack

# PSHY

**Operation:**  $(SP) - \$0002 \Rightarrow SP$   
 $(Y_H : Y_L) \Rightarrow M(SP) : M(SP + 1)$

**Description:** Stacks the content of index register Y. The stack pointer is decremented by two. The content of Y is then stacked at the address to which the SP points. After PSHY executes, the SP points to the stacked value of the high-order half of Y.

Push instructions are commonly used to save the contents of one or more CPU registers at the start of a subroutine. Complementary pull instructions can be used to restore the saved CPU registers just before returning from the subroutine.

**Condition Codes and Boolean Formulas:**

S	X	H	I	N	Z	V	C
-	-	-	-	-	-	-	-

None affected.

**Addressing Modes, Machine Code, and Execution Times:**

Source Form	Address Mode	Object Code	Cycles	Access Detail
PSHY	INH	35	2	OS

# PULA

Pull A from Stack

# PULA

**Operation:**  $M_{(SP)} \Rightarrow A$   
 $(SP) + \$0001 \Rightarrow SP$

**Description:** Accumulator A is loaded from the address indicated by the stack pointer. The SP is then incremented by one.

Pull instructions are commonly used at the end of a subroutine, to restore the contents of CPU registers that were pushed onto the stack before subroutine execution.

**Condition Codes and Boolean Formulas:**

S	X	H	I	N	Z	V	C
-	-	-	-	-	-	-	-

None affected.

**Addressing Modes, Machine Code, and Execution Times:**

Source Form	Address Mode	Object Code	Cycles	Access Detail
PULA	INH	32	3	ufo

# PULB

Pull B from Stack

# PULB

**Operation:**  $(M_{(SP)}) \Rightarrow B$   
 $(SP) + \$0001 \Rightarrow SP$

**Description:** Accumulator B is loaded from the address indicated by the stack pointer. The SP is then incremented by one.

Pull instructions are commonly used at the end of a subroutine, to restore the contents of CPU registers that were pushed onto the stack before subroutine execution.

**Condition Codes and Boolean Formulas:**

S	X	H	I	N	Z	V	C
-	-	-	-	-	-	-	-

None affected.

**Addressing Modes, Machine Code, and Execution Times:**

Source Form	Address Mode	Object Code	Cycles	Access Detail
PULB	INH	33	3	uFO



# PULC

## Pull Condition Code Register from Stack

# PULC

**Operation:**  $(M_{(SP)}) \Rightarrow CCR$   
 $(SP) + \$0001 \Rightarrow SP$

**Description:** The condition code register is loaded from the address indicated by the stack pointer. The SP is then incremented by one.

Pull instructions are commonly used at the end of a subroutine, to restore the contents of CPU registers that were pushed onto the stack before subroutine execution.

### Condition Codes and Boolean Formulas:

S	X	H	I	N	Z	V	C
$\Delta$	$\Downarrow$	$\Delta$	$\Delta$	$\Delta$	$\Delta$	$\Delta$	$\Delta$

Condition codes take on the value pulled from the stack, except that the X mask bit cannot change from zero to one. Software can leave the X bit set, leave it cleared, or change it from one to zero, but it can only be set by a reset or by recognition of an XIRQ interrupt.

### Addressing Modes, Machine Code, and Execution Times:

Source Form	Address Mode	Object Code	Cycles	Access Detail
PULC	INH	38	3	uFO

# PULD

Pull Double Accumulator  
from Stack

# PULD

**Operation:**  $(M_{(SP)} : M_{(SP + 1)}) \Rightarrow A : B$   
 $(SP) + \$0002 \Rightarrow SP$

**Description:** Double accumulator D is loaded from the address indicated by the stack pointer. The SP is then incremented by two.

The order in which A and B are pulled from the stack is the opposite of the order in which A and B are pushed when an RTI instruction is executed. The interrupt stacking order for A and B is backward-compatible with the M6800, which had no 16-bit accumulator.

Pull instructions are commonly used at the end of a subroutine, to restore the contents of CPU registers that were pushed onto the stack before subroutine execution.

**Condition Codes and Boolean Formulas:**

S	X	H	I	N	Z	V	C
-	-	-	-	-	-	-	-

None affected.

**Addressing Modes, Machine Code, and Execution Times:**

Source Form	Address Mode	Object Code	Cycles	Access Detail
PULD	INH	3A	3	UFO

# PULX

Pull Index Register X from Stack

# PULX

**Operation:**  $(M_{(SP)} : M_{(SP + 1)}) \Rightarrow X_H : X_L$   
 $(SP) + \$0002 \Rightarrow SP$

**Description:** Index register X is loaded from the address indicated by the stack pointer. The SP is then incremented by two.

Pull instructions are commonly used at the end of a subroutine, to restore the contents of CPU registers that were pushed onto the stack before subroutine execution.

**Condition Codes and Boolean Formulas:**

S	X	H	I	N	Z	V	C
—	—	—	—	—	—	—	—

None affected.

**Addressing Modes, Machine Code, and Execution Times:**

Source Form	Address Mode	Object Code	Cycles	Access Detail
PULX	INH	30	3	UfO

# PULY

## Pull Index Register Y from Stack

# PULY

**Operation:**  $(M(SP) : M(SP + 1)) \Rightarrow Y_H : Y_L$   
 $(SP) + \$0002 \Rightarrow SP$

**Description:** Index register Y is loaded from the address indicated by the stack pointer. The SP is then incremented by two.

Pull instructions are commonly used at the end of a subroutine, to restore the contents of CPU registers that were pushed onto the stack before subroutine execution.

**Condition Codes and Boolean Formulas:**

S	X	H	I	N	Z	V	C
-	-	-	-	-	-	-	-

None affected.

**Addressing Modes, Machine Code, and Execution Times:**

Source Form	Address Mode	Object Code	Cycles	Access Detail
PULY	INH	31	3	UFO

# REV

## Fuzzy Logic Rule Evaluation

# REV

**Operation:** MIN – MAX Rule Evaluation

**Description:** Performs an unweighted evaluation of a list of rules, using fuzzy input values to produce fuzzy outputs. REV can be interrupted, so it does not adversely affect interrupt latency.

The REV instruction uses an 8-bit offset from a base address stored in index register Y to determine the address of each fuzzy input and fuzzy output. For REV to execute correctly, each rule in the knowledge base must consist of a table of 8-bit antecedent offsets followed by a table of 8-bit consequent offsets. The value \$FE marks boundaries between antecedents and consequents, and between successive rules. The value \$FF marks the end of the rule list. REV can evaluate any number of rules with any number of inputs and outputs.

Beginning with the address pointed to by the first rule antecedent, REV evaluates each successive fuzzy input value until it encounters an \$FE separator. Operation is similar to that of a MINA instruction. The smallest input value is the truth value of the rule. Then, beginning with the address pointed to by the first rule consequent, the truth value is compared to each successive fuzzy output value until another \$FE separator is encountered; if the truth value is greater than the current output value, it is written to the output. Operation is similar to that of a MAXM instruction. Rules are processed until an \$FF terminator is encountered.

Before executing REV, perform the following set up operations.

- X must point to the first 8-bit element in the rule list.

- Y must point to the base address for fuzzy inputs and fuzzy outputs.

- A must contain the value \$FF, and the CCR V bit must = 0 (LDAA #\$FF places the correct value in A and clears V).

- Clear fuzzy outputs to zeros.

Index register X points to the element in the rule list that is being evaluated. X is automatically updated so that execution can resume correctly if the instruction is interrupted. When execution is complete, X points to the next address after the \$FF separator at the end of the rule list.

Index register Y points to the base address for the fuzzy inputs and fuzzy outputs. The value in Y does not change during execution.

Accumulator A holds intermediate results. During antecedent processing, a MIN function compares each fuzzy input to the value stored in A, and writes the smaller of the two to A. When all antecedents have been evaluated, A contains the smallest input value. This is the truth value used during consequent processing. Accumulator A must be initialized to \$FF for the MIN function to evaluate the inputs of the first rule correctly. For subsequent rules, the value \$FF is written to A when an \$FE marker is encountered. At the end of execution, accumulator A holds the truth value for the last rule.

The V status bit signals whether antecedents (0) or consequents (1) are being processed. V must be initialized to zero in order for processing to begin with the antecedents of the first rule. Once execution begins, the value of V is automatically changed as \$FE separators are encountered. At the end of execution, V should equal one, because the last element before the \$FF end marker should be a rule consequent. If V is equal to zero at the end of execution, the rule list is incorrect.

Fuzzy outputs must be cleared to \$00 before processing begins in order for the MAX algorithm used during consequent processing to work correctly. Residual output values would cause incorrect comparison.

Refer to **SECTION 9 FUZZY LOGIC SUPPORT** for details.

#### Condition Codes and Boolean Formulas:

S	X	H	I	N	Z	V	C
-	-	?	-	?	?	Δ	?

V: 1; Normally set, unless rule structure is erroneous.

H, N, Z and C may be altered by this instruction.

#### Addressing Modes, Machine Code, and Execution Times:

Source Form	Address Mode	Object Code	Cycles	Access Detail
REV (add if interrupted)	Special	18 3A	see note <sup>1</sup>	Orf (ttx) 0 ff + Orf

Notes:

1. The 3-cycle loop in parentheses is executed once for each element in the rule list. When an interrupt occurs, there is a 2-cycle exit sequence, a 4-cycle re-entry sequence, then execution resumes with a prefetch of the last antecedent or consequent being processed at the time of the interrupt.

# RE VW Fuzzy Logic Rule Evaluation (Weighted) RE VW

**Operation:** MIN – MAX Rule Evaluation with Optional Rule Weighting

**Description:** RE VW performs either weighted or unweighted evaluation of a list of rules, using fuzzy inputs to produce fuzzy outputs. RE VW can be interrupted, so it does not adversely affect interrupt latency.

For RE VW to execute correctly, each rule in the knowledge base must consist of a table of 16-bit antecedent pointers followed by a table of 16-bit consequent pointers. The value \$FFFE marks boundaries between antecedents and consequents, and between successive rules. The value \$FFFF marks the end of the rule list. RE VW can evaluate any number of rules with any number of inputs and outputs.

Setting the C status bit enables weighted evaluation. To use weighted evaluation, a table of 8-bit weighting factors, one per rule, must be stored in memory. Index register Y points to the weighting factors.

Beginning with the address pointed to by the first rule antecedent, RE VW evaluates each successive fuzzy input value until it encounters an \$FFFE separator. Operation is similar to that of a MINA instruction. The smallest input value is the truth value of the rule. Next, if weighted evaluation is enabled, a computation is performed, and the truth value is modified. Then, beginning with the address pointed to by the first rule consequent, the truth value is compared to each successive fuzzy output value until another \$FFFE separator is encountered; if the truth value is greater than the current output value, it is written to the output. Operation is similar to that of a MAXM instruction. Rules are processed until an \$FFFF terminator is encountered.

Perform these set up operations before execution.

- X must point to the first 16-bit element in the rule list.

- A must contain the value \$FF, and the CCR V bit must = 0 (LDAA #\$FF places the correct value in A and clears V).

- Clear fuzzy outputs to zeros.

- Set or clear the CCR C bit. When weighted evaluation is enabled, Y must point to the first item in a table of 8-bit weighting factors.

Index register X points to the element in the rule list that is being evaluated. X is automatically updated so that execution can resume correctly if the instruction is interrupted. When execution is complete, X points to the address after the \$FFFF separator at the end of the rule list.

Index register Y points to the weighting factor being used. Y is automatically updated so that execution can resume correctly if the instruction is interrupted. When execution is complete, Y points to the last weighting factor used. When weighting is not used (C = 0), Y is not changed.

Accumulator A holds intermediate results. During antecedent processing, a MIN function compares each fuzzy input to the value stored in A, and writes the smaller of the two to A. When all antecedents have been evaluated, A contains the smallest input value. For unweighted evaluation, this is the truth value used during consequent processing. For weighted evaluation, the value in A is multiplied by the quantity (Rule Weight + 1) and the upper eight bits of the result replace the content of A. Accumulator A must be initialized to \$FF for the MIN function to evaluate the inputs of the first rule correctly. For subsequent rules, the value \$FF is written to A when an \$FFFE marker is encountered. At the end of execution, accumulator A holds the truth value for the last rule.

The V status bit signals whether antecedents (0) or consequents (1) are being processed. V must be initialized to zero in order for processing to begin with the antecedents of the first rule. Once execution begins, the value of V is automatically changed as \$FFFE separators are encountered. At the end of execution, V should equal one, because the last element before the \$FF end marker should be a rule consequent. If V is equal to zero at the end of execution, the rule list is incorrect.

Fuzzy outputs must be cleared to \$00 before processing begins in order for the MAX algorithm used during consequent processing to work correctly. Residual output values would cause incorrect comparison.

Refer to **SECTION 9 FUZZY LOGIC SUPPORT** for details.

#### Condition Codes and Boolean Formulas:

S	X	H	I	N	Z	V	C
-	-	?	-	?	?	Δ	!

V: 1; Normally set, unless rule structure is erroneous.

C: Selects weighted (1) or unweighted (0) rule evaluation.

H, N, Z and C may be altered by this instruction.

#### Addressing Modes, Machine Code, and Execution Times:

Source Form	Address Mode	Object Code	Cycles	Access Detail
REVW (add 2 at end of ins if wts) (add if interrupted)	Special	18 3B	See note <sup>1</sup>	ORf (tTx)O (rffrf) fff + ORft

##### Notes:

1. The 3-cycle loop in parentheses expands to five cycles for separators when weighting is enabled. The loop is executed once for each element in the rule list. When an interrupt occurs, there is a 2-cycle exit sequence, a 4-cycle re-entry sequence, then execution resumes with a prefetch of the last antecedent or consequent being processed at the time of the interrupt.

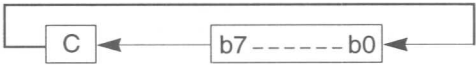


# ROL

## Rotate Left Memory

# ROL

### Operation:



**Description:** Shifts all bits of memory location M one place to the left. Bit 0 is loaded from the C status bit. The C bit is loaded from the most significant bit of M. Rotate operations include the carry bit to allow extension of shift and rotate operations to multiple bytes. For example, to shift a 24-bit value one bit to the left, the sequence ASL LOW, ROL MID, ROL HIGH could be used where LOW, MID and HIGH refer to the low-order, middle and high-order bytes of the 24-bit value, respectively.

### Condition Codes and Boolean Formulas:

S	X	H	I	N	Z	V	C
—	—	—	—	Δ	Δ	Δ	Δ

- N: Set if MSB of result is set; cleared otherwise.
- Z: Set if result is \$00; cleared otherwise.
- V:  $N \oplus C = [N \bullet \overline{C}] + [\overline{N} \bullet C]$  (for N and C after the shift)  
Set if (N is set and C is cleared) or (N is cleared and C is set); cleared otherwise (for values of N and C after the shift).
- C: M7  
Set if the MSB of M was set before the shift; cleared otherwise.

### Addressing Modes, Machine Code, and Execution Times:

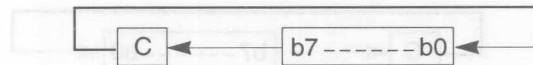
Source Form	Address Mode	Object Code	Cycles	Access Detail
ROL <i>opr16a</i>	EXT	75 hh ll	4	rOPw
ROL <i>opr0_xysp</i>	IDX	65 xb	3	rPw
ROL <i>opr9_xysp</i>	IDX1	65 xb ff	4	rPOw
ROL <i>opr16_xysp</i>	IDX2	65 xb ee ff	5	frPPw
ROL [D, <i>xysp</i> ]	[D,IDX]	65 xb	6	fIfrPw
ROL [ <i>opr16_xysp</i> ]	[IDX2]	65 xb ee ff	6	fIPrPw

# ROLA

Rotate Left A

# ROLA

## Operation:



**Description:** Shifts all bits of accumulator A one place to the left. Bit 0 is loaded from the C status bit. The C bit is loaded from the most significant bit of A. Rotate operations include the carry bit to allow extension of shift and rotate operations to multiple bytes. For example, to shift a 24-bit value one bit to the left, the sequence ASL LOW, ROL MID, ROL HIGH could be used where LOW, MID and HIGH refer to the low-order, middle and high-order bytes of the 24-bit value, respectively.

## Condition Codes and Boolean Formulas:

S	X	H	I	N	Z	V	C
-	-	-	-	Δ	Δ	Δ	Δ

N: Set if MSB of result is set; cleared otherwise.

Z: Set if result is \$00; cleared otherwise.

V:  $N \oplus C = [N \cdot \overline{C}] + [\overline{N} \cdot C]$  (for N and C after the shift)

Set if (N is set and C is cleared) or (N is cleared and C is set); cleared otherwise (for values of N and C after the shift).

C: A7

Set if the MSB of A was set before the shift; cleared otherwise.

## Addressing Modes, Machine Code, and Execution Times:

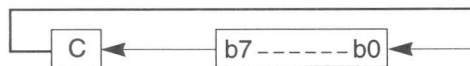
Source Form	Address Mode	Object Code	Cycles	Access Detail
ROLA	INH	45	1	0

# ROLB

Rotate Left B

# ROLB

## Operation:



**Description:** Shifts all bits of accumulator B one place to the left. Bit 0 is loaded from the C status bit. The C bit is loaded from the most significant bit of B. Rotate operations include the carry bit to allow extension of shift and rotate operations to multiple bytes. For example, to shift a 24-bit value one bit to the left, the sequence ASL LOW, ROL MID, ROL HIGH could be used where LOW, MID and HIGH refer to the low-order, middle and high-order bytes of the 24-bit value, respectively.

## Condition Codes and Boolean Formulas:

S	X	H	I	N	Z	V	C
—	—	—	—	Δ	Δ	Δ	Δ

N: Set if MSB of result is set; cleared otherwise.

Z: Set if result is \$00; cleared otherwise.

V:  $N \oplus C = [N \bullet \overline{C}] + [\overline{N} \bullet C]$  (for N and C after the shift)  
Set if (N is set and C is cleared) or (N is cleared and C is set); cleared otherwise (for values of N and C after the shift).

C: B7  
Set if the MSB of B was set before the shift; cleared otherwise.

## Addressing Modes, Machine Code, and Execution Times:

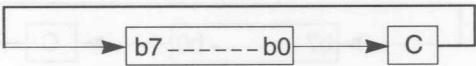
Source Form	Address Mode	Object Code	Cycles	Access Detail
ROLB	INH	55	1	0

# ROR

Rotate Right Memory

# ROR

Operation:



**Description:** Shifts all bits of memory location M one place to the right. Bit 7 is loaded from the C status bit. The C bit is loaded from the least significant bit of M. Rotate operations include the carry bit to allow extension of shift and rotate operations to multiple bytes. For example, to shift a 24-bit value one bit to the right, the sequence LSR HIGH, ROR MID, ROR LOW could be used where LOW, MID and HIGH refer to the low-order, middle, and high-order bytes of the 24-bit value, respectively.

Condition Codes and Boolean Formulas:

S	X	H	I	N	Z	V	C
-	-	-	-	$\Delta$	$\Delta$	$\Delta$	$\Delta$

- N: Set if MSB of result is set; cleared otherwise.
- Z: Set if result is \$00; cleared otherwise.
- V:  $N \oplus C = [N \bullet \bar{C}] + [\bar{N} \bullet C]$  (for N and C after the shift)  
Set if (N is set and C is cleared) or (N is cleared and C is set); cleared otherwise (for values of N and C after the shift).
- C: M0  
Set if the LSB of M was set before the shift; cleared otherwise.

Addressing Modes, Machine Code, and Execution Times:

Source Form	Address Mode	Object Code	Cycles	Access Detail
ROR <i>opr16a</i>	EXT	76 hh ll	4	rOPw
ROR <i>opr0_xysp</i>	IDX	66 xb	3	rPw
ROR <i>opr9,xysp</i>	IDX1	66 xb ff	4	rPOw
ROR <i>opr16,xysp</i>	IDX2	66 xb ee ff	5	frPPw
ROR [D, <i>xysp</i> ]	[D,IDX]	66 xb	6	fIfxPw
ROR [ <i>opr16,xysp</i> ]	[IDX2]	66 xb ee ff	6	fIPrPw

# RORA

Rotate Right A

# RORA

## Operation:



**Description:** Shifts all bits of accumulator A one place to the right. Bit 7 is loaded from the C status bit. The C bit is loaded from the least significant bit of A. Rotate operations include the carry bit to allow extension of shift and rotate operations to multiple bytes. For example, to shift a 24-bit value one bit to the right, the sequence LSR HIGH, ROR MID, ROR LOW could be used where LOW, MID and HIGH refer to the low-order, middle, and high-order bytes of the 24-bit value, respectively.

## Condition Codes and Boolean Formulas:

S	X	H	I	N	Z	V	C
—	—	—	—	Δ	Δ	Δ	Δ

N: Set if MSB of result is set; cleared otherwise.

Z: Set if result is \$00; cleared otherwise.

V:  $N \oplus C = [N \bullet \overline{C}] + [\overline{N} \bullet C]$  (for N and C after the shift)  
Set if (N is set and C is cleared) or (N is cleared and C is set); cleared otherwise (for values of N and C after the shift).

C: A0  
Set if the LSB of A was set before the shift; cleared otherwise.

## Addressing Modes, Machine Code, and Execution Times:

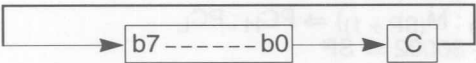
Source Form	Address Mode	Object Code	Cycles	Access Detail
RORA	INH	46	1	0

# RORB

Rotate Right B

# RORB

**Operation:**



**Description:** Shifts all bits of accumulator B one place to the right. Bit 7 is loaded from the C status bit. The C bit is loaded from the least significant bit of B. Rotate operations include the carry bit to allow extension of shift and rotate operations to multiple bytes. For example, to shift a 24-bit value one bit to the right, the sequence LSR HIGH, ROR MID, ROR LOW could be used where LOW, MID and HIGH refer to the low-order, middle and high-order bytes of the 24-bit value, respectively.

**Condition Codes and Boolean Formulas:**

S	X	H	I	N	Z	V	C
-	-	-	-	Δ	Δ	Δ	Δ

- N: Set if MSB of result is set; cleared otherwise.
- Z: Set if result is \$00; cleared otherwise.
- V:  $N \oplus C = [N \bullet C] + [\bar{N} \bullet C]$  (for N and C after the shift)  
Set if (N is set and C is cleared) or (N is cleared and C is set); cleared otherwise (for values of N and C after the shift).
- C: B0  
Set if the LSB of B was set before the shift; cleared otherwise.

**Addressing Modes, Machine Code, and Execution Times:**

Source Form	Address Mode	Object Code	Cycles	Access Detail
RORB	INH	56	1	0

# RTC

## Return from Call

# RTC

**Operation:**  $(M_{(SP)}) \Rightarrow \text{PPAGE}$   
 $(SP) + \$0001 \Rightarrow SP$   
 $(M_{(SP)} : M_{(SP + 1)}) \Rightarrow PC_H : PC_L$   
 $(SP) + \$0002 \Rightarrow SP$

**Description:** Terminates subroutines in expanded memory invoked by the CALL instruction. Returns execution flow from the subroutine to the calling program. The program overlay page (PPAGE) register and the return address are restored from the stack; program execution continues at the restored address. For code compatibility purposes, CALL and RTC also execute correctly in M68HC12 devices that do not have expanded memory capability.

### Condition Codes and Boolean Formulas:

S	X	H	I	N	Z	V	C
—	—	—	—	—	—	—	—

None affected.

### Addressing Modes, Machine Code, and Execution Times:

Source Form	Address Mode	Object Code	Cycles	Access Detail
RTC	INH	0A	6	uUnPPP

# RTI

## Return from Interrupt

# RTI

**Operation:**  $(M(SP)) \Rightarrow CCR; (SP) + \$0001 \Rightarrow SP$   
 $(M(SP) : M(SP + 1)) \Rightarrow B : A; (SP) + \$0002 \Rightarrow SP$   
 $(M(SP) : M(SP + 1)) \Rightarrow X_H : X_L; (SP) + \$0004 \Rightarrow SP$   
 $(M(SP) : M(SP + 1)) \Rightarrow PC_H : PC_L; (SP) - \$0002 \Rightarrow SP$   
 $(M(SP) : M(SP + 1)) \Rightarrow Y_H : Y_L; (SP) + \$0004 \Rightarrow SP$

**Description:** Restores system context after interrupt service processing is completed. The condition codes, accumulators B and A, index register X, the PC, and index register Y are restored to a state pulled from the stack. The X mask bit may be cleared as a result of an RTI instruction, but cannot be set if it was cleared prior to execution of the RTI instruction.

If another interrupt is pending when RTI has finished restoring registers from the stack, the SP is adjusted to preserve stack content, and the new vector is fetched. This operation is functionally identical to the same operation in the M68HC11, where registers actually are re-stacked, but is faster.

### Condition Codes and Boolean Formulas:

S	X	H	I	N	Z	V	C
$\Delta$	$\downarrow$	$\Delta$	$\Delta$	$\Delta$	$\Delta$	$\Delta$	$\Delta$

Condition codes take on the value pulled from the stack, except that the X mask bit cannot change from zero to one. Software can leave the X bit set, leave it cleared, or change it from one to zero, but it can only be set by a reset or by recognition of an  $\overline{XIRQ}$  interrupt.

### Addressing Modes, Machine Code, and Execution Times:

Source Form	Address Mode	Object Code	Cycles	Access Detail
RTI	INH	0B	8	UUUUUPPP
(with interrupt pending)			10	UUUUUVPPPP



# RTS

## Return from Subroutine

# RTS

**Operation:**  $(M_{(SP)} : M_{(SP + 1)}) \Rightarrow PC_H : PC_L; (SP) + \$0002 \Rightarrow SP$

**Description:** Restores context at the end of a subroutine. Loads the program counter with a 16-bit value pulled from the stack and increments the stack pointer by two. Program execution continues at the address restored from the stack.

**Condition Codes and Boolean Formulas:**

S	X	H	I	N	Z	V	C
–	–	–	–	–	–	–	–

None affected.

**Addressing Modes, Machine Code, and Execution Times:**

Source Form	Address Mode	Object Code	Cycles	Access Detail
RTS	INH	3D	5	UÉPPP

# SBA

## Subtract Accumulators

# SBA

**Operation:**  $(A) - (B) \Rightarrow A$

**Description:** Subtracts the content of accumulator B from the content of accumulator A and places the result in A. The content of B is not affected. For subtraction instructions, the C status bit represents a borrow.

**Condition Codes and Boolean Formulas:**

S	X	H	I	N	Z	V	C
-	-	-	-	$\Delta$	$\Delta$	$\Delta$	$\Delta$

- N: Set if MSB of result is set; cleared otherwise.
- Z: Set if result is \$00; cleared otherwise.
- V:  $A7 \bullet B7 \bullet R7 + \overline{A7} \bullet \overline{B7} \bullet \overline{R7}$   
Set if a two's complement overflow resulted from the operation; cleared otherwise.
- C:  $\overline{A7} \bullet B7 + B7 \bullet R7 + R7 \bullet \overline{A7}$   
Set if the absolute value of B is larger than the absolute value of A; cleared otherwise.

**Addressing Modes, Machine Code, and Execution Times:**

Source Form	Address Mode	Object Code	Cycles	Access Detail
SBA	INH	18 16	2	00

# SBCA

Subtract with Carry from A

# SBCA

**Operation:**  $(A) - (M) - C \Rightarrow A$

**Description:** Subtracts the content of memory location M and the value of the C status bit from the content of accumulator A. The result is placed in A. For subtraction instructions, the C status bit represents a borrow.

## Condition Codes and Boolean Formulas:

S	X	H	I	N	Z	V	C
—	—	—	—	Δ	Δ	Δ	Δ

N: Set if MSB of result is set; cleared otherwise.

Z: Set if result is \$00; cleared otherwise.

V:  $X7 \bullet \overline{M7} \bullet \overline{R7} + \overline{X7} \bullet M7 \bullet R7$   
Set if a two's complement overflow resulted from the operation; cleared otherwise.

C:  $\overline{X7} \bullet M7 + M7 \bullet R7 + R7 \bullet \overline{X7}$   
Set if the absolute value of the content of memory plus previous carry is larger than the absolute value of the accumulator; cleared otherwise.

## Addressing Modes, Machine Code, and Execution Times:

Source Form	Address Mode	Object Code	Cycles	Access Detail
SBCA #opr8i	IMM	82 ii	1	P
SBCA opr8a	DIR	92 dd	3	rfrP
SBCA opr16a	EXT	B2 hh ll	3	rOP
SBCA oprx0,xysp	IDX	A2 xb	3	rfrP
SBCA oprx9,xysp	IDX1	A2 xb ff	3	rPO
SBCA oprx16,xysp	IDX2	A2 xb ee ff	4	frPP
SBCA [D,xysp]	[D,IDX]	A2 xb	6	frfrfrP
SBCA [oprx16,xysp]	[IDX2]	A2 xb ee ff	6	frPrfrP

# SBCB

Subtract with Carry from B

# SBCB

**Operation:**  $(B) - (M) - C \Rightarrow B$

**Description:** Subtracts the content of memory location M and the value of the C status bit from the content of accumulator B. The result is placed in B. For subtraction instructions, the C status bit represents a borrow.

**Condition Codes and Boolean Formulas:**

S	X	H	I	N	Z	V	C
-	-	-	-	$\Delta$	$\Delta$	$\Delta$	$\Delta$

N: Set if MSB of result is set; cleared otherwise.

Z: Set if result is \$00; cleared otherwise.

V:  $X7 \bullet \overline{M7} \bullet \overline{R7} + \overline{X7} \bullet M7 \bullet R7$

Set if a two's complement overflow resulted from the operation; cleared otherwise.

C:  $\overline{X7} \bullet M7 + M7 \bullet R7 + R7 \bullet \overline{X7}$

Set if the absolute value of the content of memory plus previous carry is larger than the absolute value of the accumulator; cleared otherwise.

**Addressing Modes, Machine Code, and Execution Times:**

Source Form	Address Mode	Object Code	Cycles	Access Detail
SBCB #opr8i	IMM	C2 ii	1	P
SBCB opr8a	DIR	D2 dd	3	rFP
SBCB opr16a	EXT	F2 hh ll	3	rOP
SBCB oprx0,xysp	IDX	E2 xb	3	rFP
SBCB oprx9,xysp	IDX1	E2 xb ff	3	rPO
SBCB oprx16,xysp	IDX2	E2 xb ee ff	4	frPP
SBCB [D,xysp]	[D,IDX]	E2 xb	6	fIfrfP
SBCB [oprx16,xysp]	[IDX2]	E2 xb ee ff	6	fIPrfP

# SEC

Set Carry

# SEC

**Operation:** 1 ⇒ C bit

**Description:** Sets the C status bit. This instruction is assembled as ORCC #\$01. The ORCC instruction can be used to set any combination of bits in the CCR in one operation.

SEC can be used to set up the C bit prior to a shift or rotate instruction involving the C bit.

**Condition Codes and Boolean Formulas:**

S	X	H	I	N	Z	V	C
—	—	—	—	—	—	—	1

C: 1; Set.

**Addressing Modes, Machine Code, and Execution Times:**

Source Form	Address Mode	Object Code	Cycles	Access Detail
SEC translates to... ORCC #\$01	IMM	14 01	1	P

# SEI

## Set Interrupt Mask

# SEI

**Operation:** 1  $\Rightarrow$  I bit

**Description:** Sets the I mask bit. This instruction is assembled as ORCC #\$10. The ORCC instruction can be used to set any combination of bits in the CCR in one operation. When the I bit is set, all maskable interrupts are inhibited, and the CPU will recognize only non-maskable interrupt sources or an SWI.

### Condition Codes and Boolean Formulas:

S	X	H	I	N	Z	V	C
-	-	-	1	-	-	-	-

I: 1; Set.

### Addressing Modes, Machine Code, and Execution Times:

Source Form	Address Mode	Object Code	Cycles	Access Detail
SEI translates to... ORCC #\$10	IMM	14 10	1	P

# SEV

Set Two's Complement Overflow Bit

# SEV

**Operation:** 1  $\Rightarrow$  V bit

**Description:** Sets the V status bit. This instruction is assembled as ORCC #\$02. The ORCC instruction can be used to set any combination of bits in the CCR in one operation.

**Condition Codes and Boolean Formulas:**

S	X	H	I	N	Z	V	C
—	—	—	—	—	—	1	—

V: 1; Set.

**Addressing Modes, Machine Code, and Execution Times:**

Source Form	Address Mode	Object Code	Cycles	Access Detail
SEV <i>translates to...</i> ORCC #\$02	IMM	14 02	1	P

# SEX

## Sign Extend into 16-bit Register

# SEX

**Operation:** If r1 bit 7 = 0, then \$00 : (r1)  $\Rightarrow$  r2  
If r1 bit 7 = 1, then \$FF : (r1)  $\Rightarrow$  r2

**Description:** This instruction is an alternate mnemonic for the TFR r1,r2 instruction, where r1 is an 8-bit register and r2 is a 16-bit register. The result in r2 is the 16-bit sign extended representation of the original two's complement number in r1. The content of r1 is unchanged in all cases except that of SEX A,D (D is A : B).

### Condition Codes and Boolean Formulas:

S	X	H	I	N	Z	V	C
-	-	-	-	-	-	-	-

None affected.

### Addressing Modes, Machine Code, and Execution Times:

Source Form	Address Mode	Object Code <sup>1</sup>	Cycles	Access Detail
SEX <i>abc,dxys</i>	INH	B7 eb	1	P

Notes:

1. Legal coding for eb is summarized in the following table. Columns represent the high-order digit, and rows represent the low-order digit in hexadecimal (MSB is a don't-care).

	0	1	2
3	sex:A $\Rightarrow$ TMP2	sex:B $\Rightarrow$ TMP2	sex:CCR $\Rightarrow$ TMP2
4	sex:A $\Rightarrow$ D SEX A,D	sex:B $\Rightarrow$ D SEX B,D	sex:CCR $\Rightarrow$ D SEX CCR,D
5	sex:A $\Rightarrow$ X SEX A,X	sex:B $\Rightarrow$ X SEX B,X	sex:CCR $\Rightarrow$ X SEX CCR,X
6	sex:A $\Rightarrow$ Y SEX A,Y	sex:B $\Rightarrow$ Y SEX B,Y	sex:CCR $\Rightarrow$ Y SEX CCR,Y
7	sex:A $\Rightarrow$ SP SEX A,SP	sex:B $\Rightarrow$ SP SEX B,SP	sex:CCR $\Rightarrow$ SP SEX CCR,SP



# STAA

Store Accumulator A

# STAA

**Operation:** (A) ⇒ M

**Description:** Stores the content of accumulator A in memory location M. The content of A is unchanged.

**Condition Codes and Boolean Formulas:**

S	X	H	I	N	Z	V	C
—	—	—	—	Δ	Δ	0	—

- N: Set if MSB of result is set; cleared otherwise.  
Z: Set if result is \$00; cleared otherwise.  
V: 0; Cleared.

**Addressing Modes, Machine Code, and Execution Times:**

Source Form	Address Mode	Object Code	Cycles	Access Detail
STAA <i>opr8a</i>	DIR	5A dd	2	Pw
STAA <i>opr16a</i>	EXT	7A hh ll	3	wOP
STAA <i>opr0_xysp</i>	IDX	6A xb	2	Pw
STAA <i>opr9,xysp</i>	IDX1	6A xb ff	3	PwO
STAA <i>opr16,xysp</i>	IDX2	6A xb ee ff	3	PwP
STAA [D, <i>xysp</i> ]	[D,IDX]	6A xb	5	PIfPw
STAA [ <i>opr16,xysp</i> ]	[IDX2]	6A xb ee ff	5	PIPPw

# STAB

Store Accumulator B

# STAB

Operation: (B) ⇒ M

Description: Stores the content of accumulator B in memory location M. The content of B is unchanged.

Condition Codes and Boolean Formulas:

S	X	H	I	N	Z	V	C
-	-	-	-	Δ	Δ	0	-

N: Set if MSB of result is set; cleared otherwise.

Z: Set if result is \$00; cleared otherwise.

V: 0; Cleared.

Addressing Modes, Machine Code, and Execution Times:

Source Form	Address Mode	Object Code	Cycles	Access Detail
STAB <i>opr8a</i>	DIR	5B dd	2	Pw
STAB <i>opr16a</i>	EXT	7B hh ll	3	wOP
STAB <i>opr0_xysp</i>	IDX	6B xb	2	Pw
STAB <i>opr9_xysp</i>	IDX1	6B xb ff	3	PwO
STAB <i>opr16_xysp</i>	IDX2	6B xb ee ff	3	PwP
STAB [D, <i>xysp</i> ]	[D,IDX]	6B xb	5	PIfPw
STAB [ <i>opr16_xysp</i> ]	[IDX2]	6B xb ee ff	5	PIPPw

# STD

## Store Double Accumulator

# STD

**Operation:**  $(A : B) \Rightarrow M : M + 1$

**Description:** Stores the content of double accumulator D in memory location M : M + 1. The content of D is unchanged.

### Condition Codes and Boolean Formulas:

S	X	H	I	N	Z	V	C
—	—	—	—	$\Delta$	$\Delta$	0	—

N: Set if MSB of result is set; cleared otherwise.

Z: Set if result is \$0000; cleared otherwise.

V: 0; Cleared.

### Addressing Modes, Machine Code, and Execution Times:

Source Form	Address Mode	Object Code	Cycles	Access Detail
STD <i>opr8a</i>	DIR	5C dd	2	PW
STD <i>opr16a</i>	EXT	7C hh ll	3	WOP
STD <i>opr0_xysp</i>	IDX	6C xb	2	PW
STD <i>opr9_xysp</i>	IDX1	6C xb ff	3	PWO
STD <i>opr16_xysp</i>	IDX2	6C xb ee ff	3	PWP
STD [D, <i>xysp</i> ]	[D,IDX]	6C xb	5	PIfPW
STD [ <i>opr16_xysp</i> ]	[IDX2]	6C xb ee ff	5	PIPPW

# STOP

## Stop Processing

# STOP

**Operation:** (SP) - \$0002  $\Rightarrow$  SP; RTN<sub>H</sub> : RTN<sub>L</sub>  $\Rightarrow$  (M<sub>(SP)</sub> : M<sub>(SP + 1)</sub>)  
(SP) - \$0002  $\Rightarrow$  SP; Y<sub>H</sub> : Y<sub>L</sub>  $\Rightarrow$  (M<sub>(SP)</sub> : M<sub>(SP + 1)</sub>)  
(SP) - \$0002  $\Rightarrow$  SP; X<sub>H</sub> : X<sub>L</sub>  $\Rightarrow$  (M<sub>(SP)</sub> : M<sub>(SP + 1)</sub>)  
(SP) - \$0002  $\Rightarrow$  SP; B : A  $\Rightarrow$  (M<sub>(SP)</sub> : M<sub>(SP + 1)</sub>)  
(SP) - \$0001  $\Rightarrow$  SP; CCR  $\Rightarrow$  (M<sub>(SP)</sub>)  
Stop All Clocks

**Description:** When the S control bit is set, STOP is disabled and operates like a two-cycle NOP instruction. When the S bit is cleared, STOP stacks CPU context, stops all system clocks, and puts the device in standby mode.

Standby operation minimizes system power consumption. The contents of registers and the states of I/O pins remain unchanged.

Asserting the  $\overline{\text{RESET}}$ ,  $\overline{\text{XIRQ}}$ , or  $\overline{\text{IRQ}}$  signals ends standby mode. Stacking on entry to STOP allows the CPU to recover quickly when an interrupt is used, provided a stable clock is applied to the device. If the system uses a clock reference crystal that also stops during low-power mode, crystal start-up delay lengthens recovery time.

If  $\overline{\text{XIRQ}}$  is asserted while the X mask bit = 0 ( $\overline{\text{XIRQ}}$  interrupts enabled), execution resumes with a vector fetch for the  $\overline{\text{XIRQ}}$  interrupt. If the X mask bit = 1 ( $\overline{\text{XIRQ}}$  interrupts disabled), a two-cycle recovery sequence including an O cycle is used to adjust the instruction queue, and execution continues with the next instruction after STOP.

### Condition Codes and Boolean Formulas:

S	X	H	I	N	Z	V	C
-	-	-	-	-	-	-	-

None affected.

### Addressing Modes, Machine Code, and Execution Times:

Source Form	Address Mode	Object Code	Cycles	Access Detail
STOP (entering STOP)	INH	18 3E	9	OOSSEfSsf
(exiting STOP)			5	VfPPP
(continue)			2	fO
(if STOP disabled)			2	OO

# STS

## Store Stack Pointer

# STS

**Operation:**  $(SP_H : SP_L) \Rightarrow M : M + 1$

**Description:** Stores the content of the stack pointer in memory. The most significant byte of the SP is stored at the specified address, and the least significant byte of the SP is stored at the next higher byte address (the specified address plus one).

**Condition Codes and Boolean Formulas:**

S	X	H	I	N	Z	V	C
—	—	—	—	$\Delta$	$\Delta$	0	—

N: Set if MSB of result is set; cleared otherwise.

Z: Set if result is \$0000; cleared otherwise.

V: 0; Cleared.

**Addressing Modes, Machine Code, and Execution Times:**

Source Form	Address Mode	Object Code	Cycles	Access Detail
STS <i>opr8a</i>	DIR	5F dd	2	PW
STS <i>opr16a</i>	EXT	7F hh ll	3	WOP
STS <i>opr0_xysp</i>	IDX	6F xb	2	PW
STS <i>opr9_xysp</i>	IDX1	6F xb ff	3	PWO
STS <i>opr16_xysp</i>	IDX2	6F xb ee ff	3	PWP
STS [D, <i>xysp</i> ]	[D,IDX]	6F xb	5	PIfPW
STS [ <i>opr16_xysp</i> ]	[IDX2]	6F xb ee ff	5	PIPPW

# STX

## Store Index Register X

# STX

**Operation:**  $(X_H : X_L) \Rightarrow M : M + 1$

**Description:** Stores the content of index register X in memory. The most significant byte of X is stored at the specified address, and the least significant byte of X is stored at the next higher byte address (the specified address plus one).

**Condition Codes and Boolean Formulas:**

S	X	H	I	N	Z	V	C
-	-	-	-	$\Delta$	$\Delta$	0	-

N: Set if MSB of result is set; cleared otherwise.

Z: Set if result is \$0000; cleared otherwise.

V: 0; Cleared.

**Addressing Modes, Machine Code, and Execution Times:**

Source Form	Address Mode	Object Code	Cycles	Access Detail
STX <i>opr8a</i>	DIR	5E dd	2	PW
STX <i>opr16a</i>	EXT	7E hh ll	3	WOP
STX <i>opr0_xysp</i>	IDX	6E xb	2	PW
STX <i>opr9_xysp</i>	IDX1	6E xb ff	3	PWO
STX <i>opr16_xysp</i>	IDX2	6E xb ee ff	3	PWP
STX [D, <i>xysp</i> ]	[D,IDX]	6E xb	5	PIfPW
STX [ <i>opr16_xysp</i> ]	[IDX2]	6E xb ee ff	5	PIPPW

# STY

## Store Index Register Y

# STY

**Operation:**  $(Y_H : Y_L) \Rightarrow M : M + 1$

**Description:** Stores the content of index register Y in memory. The most significant byte of Y is stored at the specified address, and the least significant byte of Y is stored at the next higher byte address (the specified address plus one).

**Condition Codes and Boolean Formulas:**

S	X	H	I	N	Z	V	C
–	–	–	–	$\Delta$	$\Delta$	0	–

N: Set if MSB of result is set; cleared otherwise.

Z: Set if result is \$0000; cleared otherwise.

V: 0; Cleared.

**Addressing Modes, Machine Code, and Execution Times:**

Source Form	Address Mode	Object Code	Cycles	Access Detail
STY <i>opr8a</i>	DIR	5D dd	2	PW
STY <i>opr16a</i>	EXT	7D hh ll	3	WOP
STY <i>opr0_xysp</i>	IDX	6D xb	2	PW
STY <i>opr9_xysp</i>	IDX1	6D xb ff	3	PWO
STY <i>opr16_xysp</i>	IDX2	6D xb ee ff	3	PWP
STY [D, <i>xysp</i> ]	[D,IDX]	6D xb	5	PIfPW
STY [ <i>opr16_xysp</i> ]	[!DX2]	6D xb ee ff	5	PIPFW

# SUBA

Subtract A

# SUBA

**Operation:**  $(A) - (M) \Rightarrow A$

**Description:** Subtracts the content of memory location M from the content of accumulator A, and places the result in A. For subtraction instructions, the C status bit represents a borrow.

**Condition Codes and Boolean Formulas:**

S	X	H	I	N	Z	V	C
-	-	-	-	$\Delta$	$\Delta$	$\Delta$	$\Delta$

N: Set if MSB of result is set; cleared otherwise.

Z: Set if result is \$00; cleared otherwise.

V:  $X7 \cdot \overline{M7} \cdot \overline{R7} + \overline{X7} \cdot M7 \cdot R7$

Set if a two's complement overflow resulted from the operation; cleared otherwise.

C:  $\overline{X7} \cdot M7 + M7 \cdot R7 + R7 \cdot \overline{X7}$

Set if the value of the content of memory is larger than the value of the accumulator; cleared otherwise.

**Addressing Modes, Machine Code, and Execution Times:**

Source Form	Address Mode	Object Code	Cycles	Access Detail
SUBA #opr8i	IMM	80 ii	1	P
SUBA opr8a	DIR	90 dd	3	rFP
SUBA opr16a	EXT	B0 hh ll	3	rOP
SUBA oprx0_xysp	IDX	A0 xb	3	rFP
SUBA oprx9_xysp	IDX1	A0 xb ff	3	rPO
SUBA oprx16_xysp	IDX2	A0 xb ee ff	4	frPP
SUBA [D,xysp]	[D,IDX]	A0 xb	6	fIfPrFP
SUBA [oprx16,xysp]	[IDX2]	A0 xb ee ff	6	fIPrFP



# SUBB

Subtract B

# SUBB

**Operation:** (B) – (M) ⇒ B

**Description:** Subtracts the content of memory location M from the content of accumulator B and places the result in B. For subtraction instructions, the C status bit represents a borrow.

**Condition Codes and Boolean Formulas:**

S	X	H	I	N	Z	V	C
–	–	–	–	Δ	Δ	Δ	Δ

- N: Set if MSB of result is set; cleared otherwise.
- Z: Set if result is \$00; cleared otherwise.
- V:  $X7 \bullet \overline{M7} \bullet R7 + \overline{X7} \bullet M7 \bullet R7$   
Set if a two's complement overflow resulted from the operation; cleared otherwise.
- C:  $\overline{X7} \bullet M7 + M7 \bullet R7 + R7 \bullet \overline{X7}$   
Set if the value of the content of memory is larger than the value of the accumulator; cleared otherwise.

**Addressing Modes, Machine Code, and Execution Times:**

Source Form	Address Mode	Object Code	Cycles	Access Detail
SUBB #opr8i	IMM	C0 ii	1	P
SUBB opr8a	DIR	D0 dd	3	rFP
SUBB opr16a	EXT	F0 hh ll	3	rOP
SUBB oprx0_xysp	IDX	E0 xb	3	rFP
SUBB oprx9_xysp	IDX1	E0 xb ff	3	rPO
SUBB oprx16_xysp	IDX2	E0 xb ee ff	4	frPP
SUBB [D,xysp]	[D,IDX]	E0 xb	6	fIfFP
SUBB [oprx16,xysp]	[IDX2]	E0 xb ee ff	6	fIPrFP

# SUBD

## Subtract Double Accumulator

# SUBD

**Operation:**  $(A : B) - (M : M + 1) \Rightarrow A : B$

**Description:** Subtracts the content of memory location  $M : M + 1$  from the content of double accumulator D and places the result in D. For subtraction instructions, the C status bit represents a borrow.

**Condition Codes and Boolean Formulas:**

S	X	H	I	N	Z	V	C
-	-	-	-	$\Delta$	$\Delta$	$\Delta$	$\Delta$

N: Set if MSB of result is set; cleared otherwise.

Z: Set if result is \$0000; cleared otherwise.

V:  $D15 \bullet \overline{M15} \bullet \overline{R15} + \overline{D15} \bullet M15 \bullet R15$

Set if a two's complement overflow resulted from the operation; cleared otherwise.

C:  $\overline{D15} \bullet M15 + M15 \bullet R15 + R15 \bullet \overline{D15}$

Set if the value of the content of memory is larger than the value of the accumulator; cleared otherwise.

**Addressing Modes, Machine Code, and Execution Times:**

Source Form	Address Mode	Object Code	Cycles	Access Detail
SUBD #opr16i	IMM	83 jj kk	2	OP
SUBD opr8a	DIR	93 dd	3	RfP
SUBD opr16a	EXT	B3 hh ll	3	ROP
SUBD oprx0_xysp	IDX	A3 xb	3	RfP
SUBD oprx9_xysp	IDX1	A3 xb ff	3	RPO
SUBD oprx16_xysp	IDX2	A3 xb ee ff	4	fRPP
SUBD [D,xysp]	[D,IDX]	A3 xb	6	fIfRfP
SUBD [oprx16,xysp]	[IDX2]	A3 xb ee ff	6	fIPRfP

# SWI

## Software Interrupt

# SWI

**Operation:**  $(SP) - \$0002 \Rightarrow SP; RTN_H : RTN_L \Rightarrow (M_{(SP)} : M_{(SP + 1)})$   
 $(SP) - \$0002 \Rightarrow SP; Y_H : Y_L \Rightarrow (M_{(SP)} : M_{(SP + 1)})$   
 $(SP) - \$0002 \Rightarrow SP; X_H : X_L \Rightarrow (M_{(SP)} : M_{(SP + 1)})$   
 $(SP) - \$0002 \Rightarrow SP; B : A \Rightarrow (M_{(SP)} : M_{(SP + 1)})$   
 $(SP) - \$0001 \Rightarrow SP; CCR \Rightarrow (M_{(SP)})$   
 $1 \Rightarrow I$   
 $(SWI\ Vector) \Rightarrow PC$

**Description:** Causes an interrupt without an external interrupt service request. Uses the address of the next instruction after SWI as a return address. Stacks the return address, index registers Y and X, accumulators B and A, and the CCR, decrementing the SP before each item is stacked. The I mask bit is then set, the PC is loaded with the SWI vector, and instruction execution resumes at that location. SWI is not affected by the I mask bit. Refer to **SECTION 7 EXCEPTION PROCESSING** for more information.

### Condition Codes and Boolean Formulas:

S	X	H	I	N	Z	V	C
-	-	-	1	-	-	-	-

I: 1; Set.

### Addressing Modes, Machine Code, and Execution Times:

Source Form	Address Mode	Object Code	Cycles	Access Detail
SWI	INH	3F	9	VSPSSPSP <sup>1</sup>

**Notes:**

1. The CPU also uses the SWI processing sequence for hardware interrupts and unimplemented opcode traps. A variation of the sequence (VFPPP) is used for resets.

# TAB

Transfer from Accumulator A  
to Accumulator B

# TAB

Operation: (A) ⇒ B

**Description:** Moves the content of accumulator A to accumulator B. The former content of B is lost; the content of A is not affected. Unlike the general transfer instruction TFR A,B which does not affect condition codes, the TAB instruction affects the N, Z, and V status bits for compatibility with M68HC11.

**Condition Codes and Boolean Formulas:**

S	X	H	I	N	Z	V	C
-	-	-	-	Δ	Δ	0	-

- N: Set if MSB of result is set; cleared otherwise.  
Z: Set if result is \$00; cleared otherwise.  
V: 0; Cleared.

**Addressing Modes, Machine Code, and Execution Times:**

Source Form	Address Mode	Object Code	Cycles	Access Detail
TAB	INH	18 0E	2	00

# TAP

## Transfer from Accumulator A to Condition Code Register

# TAP

**Operation:** (A)  $\Rightarrow$  CCR

**Description:** Transfers the logic states of bits [7:0] of accumulator A to the corresponding bit positions of the CCR. The content of A remains unchanged. The X mask bit can be cleared as a result of a TAP, but cannot be set if it was cleared prior to execution of the TAP. If the I bit is cleared, there is a one cycle delay before the system allows interrupt requests. This prevents interrupts from occurring between instructions in the sequences CLI, WAI and CLI, SEI.

This instruction is accomplished with the TFR A,CCR instruction. For compatibility with the M68HC11, the mnemonic TAP is translated by the assembler.

### Condition Codes and Boolean Formulas:

S	X	H	I	N	Z	V	C
$\Delta$	$\Downarrow$	$\Delta$	$\Delta$	$\Delta$	$\Delta$	$\Delta$	$\Delta$

Condition codes take on the value of the corresponding bit of accumulator A, except that the X mask bit cannot change from zero to one. Software can leave the X bit set, leave it cleared, or change it from one to zero, but it can only be set by a reset or by recognition of an  $\overline{XIRQ}$  interrupt.

### Addressing Modes, Machine Code, and Execution Times:

Source Form	Address Mode	Object Code	Cycles	Access Detail
TAP translates to... TFR A,CCR	INH	B7 02	1	P

# TBA

## Transfer from Accumulator B to Accumulator A

# TBA

**Operation:** (B) ⇒ A

**Description:** Moves the content of accumulator B to accumulator A. The former content of A is lost; the content of B is not affected. Unlike the general transfer instruction TFR B,A, which does not affect condition codes, the TBA instruction affects N, Z, and V for compatibility with M68HC11.

### Condition Codes and Boolean Formulas:

S	X	H	I	N	Z	V	C
-	-	-	-	Δ	Δ	0	-

N: Set if MSB of result is set; cleared otherwise.

Z: Set if result is \$00; cleared otherwise.

V: 0; Cleared.

### Addressing Modes, Machine Code, and Execution Times:

Source Form	Address Mode	Object Code	Cycles	Access Detail
TBA	INH	18 0F	2	00

Source Form	Address Mode	Object Code	Cycles	Access Detail
TBA	INH	18 0F	2	00

For more information on the TBA instruction, see the M68HC11 instruction set manual, M68HC11-1, or the M68HC11 instruction set manual, M68HC11-2.

Object Code	Source Form	Object Code	Source Form	Object Code	Source Form
00	TBA	01	TBA	02	TBA
01	TBA	03	TBA	04	TBA
02	TBA	05	TBA	06	TBA
03	TBA	07	TBA	08	TBA
04	TBA	09	TBA	0A	TBA
05	TBA	0B	TBA	0C	TBA
06	TBA	0D	TBA	0E	TBA
07	TBA	0F	TBA	10	TBA
08	TBA	11	TBA	12	TBA

# TBEQ

Test and Branch if Equal to Zero

# TBEQ

**Operation:** If (Counter) = 0, then (PC) + \$0003 + Rel  $\Rightarrow$  PC

**Description:** Tests the specified counter register A, B, D, X, Y, or SP. If the counter register is zero, branches to the specified relative destination. TBEQ is encoded into three bytes of machine code including a 9-bit relative offset (–256 to +255 locations from the start of the next instruction).

DBEQ and IBEQ instructions are similar to TBEQ, except that the counter is decremented or incremented rather than simply being tested. Bits 7 and 6 of the instruction postbyte are used to determine which operation is to be performed.

## Condition Codes and Boolean Formulas:

S	X	H	I	N	Z	V	C
–	–	–	–	–	–	–	–

None affected.

## Addressing Modes, Machine Code, and Execution Times:

Source Form	Address Mode	Object Code <sup>1</sup>	Cycles	Access Detail
TBEQ <i>abdxys,rel9</i>	REL	04 1b rr	3/3	PPP

Notes:

1. Encoding for 1b is summarized in the following table. Bit 3 is not used (don't care), bit 5 selects branch on zero (TBEQ – 0) or not zero (TBNE – 1) versions, and bit 4 is the sign bit of the 9-bit relative offset. Bits 7 and 6 should be 0:1 for TBEQ.

Count Register	Bits 2:0	Source Form	Object Code (if offset is positive)	Object Code (if offset is negative)
A	000	TBEQ A, <i>rel9</i>	04 40 rr	04 50 rr
B	001	TBEQ B, <i>rel9</i>	04 41 rr	04 51 rr
D	100	TBEQ D, <i>rel9</i>	04 44 rr	04 54 rr
X	101	TBEQ X, <i>rel9</i>	04 45 rr	04 55 rr
Y	110	TBEQ Y, <i>rel9</i>	04 46 rr	04 56 rr
SP	111	TBEQ SP, <i>rel9</i>	04 47 rr	04 57 rr

# TBL

## Table Lookup and Interpolate

# TBL

**Operation:**  $(M) + [(B) \times ((M+1) - (M))] \Rightarrow A$

**Description:** Linearly interpolates one of 256 result values that fall between each pair of data entries in a lookup table stored in memory. Data points in the table represent the endpoints of equally spaced line segments. Table entries and the interpolated result are 8-bit values. The result is stored in accumulator A.

Before executing TBL, set up an index register so that it will point to the starting point (X1) of a line segment when the instruction is executed. X1 is the table entry closest to, but less than or equal to, the desired lookup value. The next table entry after X1 is X2. XL is the X position of the desired lookup point. Load accumulator B with a binary fraction (radix point to left of MSB), representing the ratio  $(XL - X1) \div (X2 - X1)$ .

The 8-bit unrounded result is calculated using the following expression:

$$A = Y1 + [(B) \times (Y2 - Y1)]$$

Where

$$(B) = (XL - X1) \div (X2 - X1)$$

Y1 = 8-bit data entry pointed to by <effective address>

Y2 = 8-bit data entry pointed to by <effective address> + 1

The intermediate value  $[(B) \times (Y2 - Y1)]$  produces a 16-bit result with the radix point between bits 7 and 8. The result in A is the upper 8-bits (integer part) of this intermediate 16-bit value, plus the 8-bit value Y1.

Any indexed addressing mode referenced to X, Y, SP, or PC, except in-direct modes or 9-bit and 16-bit offset modes, can be used to identify the first data point (X1,Y1). The second data point is the next table entry.

### Condition Codes and Boolean Formulas:

S	X	H	I	N	Z	V	C
-	-	-	-	Δ	Δ	-	?

N: Set if MSB of result is set; cleared otherwise.

Z: Set if result is \$00; cleared otherwise.

C: Undefined.

### Addressing Modes, Machine Code, and Execution Times:

Source Form	Address Mode	Object Code	Cycles	Access Detail
TBL <i>opr0_xysp</i>	IDX	18 3D xb	8	OrrffffP



# TBNE

Test and Branch if Not Equal to Zero

# TBNE

**Operation:** If (Counter)  $\neq 0$ , then (PC) + \$0003 + Rel  $\Rightarrow$  PC,

**Description:** Tests the specified counter register A, B, D, X, Y, or SP. If the counter register is not zero, branches to the specified relative destination. TBNE is encoded into three bytes of machine code including a 9-bit relative offset (–256 to +255 locations from the start of the next instruction).

DBNE and IBNE instructions are similar to TBNE, except that the counter is decremented or incremented rather than simply being tested. Bits 7 and 6 of the instruction postbyte are used to determine which operation is to be performed.

## Condition Codes and Boolean Formulas:

S	X	H	I	N	Z	V	C
–	–	–	–	–	–	–	–

None affected.

## Addressing Modes, Machine Code, and Execution Times:

Source Form	Address Mode	Object Code <sup>1</sup>	Cycles	Access Detail
TBNE <i>abdxys,rel9</i>	REL	04 1b rr	3/3	PPP

Notes:

1. Encoding for 1b is summarized in the following table. Bit 3 is not used (don't care), bit 5 selects branch on zero (TBEQ – 0) or not zero (TBNE – 1) versions, and bit 4 is the sign bit of the 9-bit relative offset. Bits 7 and 6 should be 0:1 for TBNE.

Count Register	Bits 2:0	Source Form	Object Code (if offset is positive)	Object Code (if offset is negative)
A	000	TBNE A, <i>rel9</i>	04 60 rr	04 70 rr
B	001	TBNE B, <i>rel9</i>	04 61 rr	04 71 rr
D	100	TBNE D, <i>rel9</i>	04 64 rr	04 74 rr
X	101	TBNE X, <i>rel9</i>	04 65 rr	04 75 rr
Y	110	TBNE Y, <i>rel9</i>	04 66 rr	04 76 rr
SP	111	TBNE SP, <i>rel9</i>	04 67 rr	04 77 rr

# TFR

## Transfer Register Content to Another Register

# TFR

**Operation:** See table.

**Description:** Transfers the content of a source register to a destination register specified in the instruction. The order in which transfers between 8-bit and 16-bit registers are specified affects the high byte of the 16-bit registers differently. Cases involving TMP2 and TMP3 are reserved for Motorola use, so some assemblers may not permit their use. It is possible to generate these cases by using DC.B or DC.W assembler directives.

### Condition Codes and Boolean Formulas:

S	X	H	I	N	Z	V	C
-	-	-	-	-	-	-	-

or

Δ	↓	Δ	Δ	Δ	Δ	Δ	Δ
---	---	---	---	---	---	---	---

None affected, unless the CCR is the destination register. Condition codes take on the value of the corresponding source bits, except that the X mask bit cannot change from zero to one. Software can leave the X bit set, leave it cleared, or change it from one to zero, but it can only be set by a reset or by recognition of an XIRQ interrupt.

### Addressing Modes, Machine Code, and Execution Times:

Source Form	Address Mode	Object Code <sup>1</sup>	Cycles	Access Detail
TFR <i>abcdxys,abcdxys</i>	INH	B7 eb	1	P

Notes:

1. Legal coding for eb is summarized in the following table. Columns represent the high-order digit, and rows represent the low-order digit in hexadecimal (MSB is a don't-care).

	0	1	2	3	4	5	6	7
0	A ⇒ A	B ⇒ A	CCR ⇒ A	TMP3 <sub>L</sub> ⇒ A	B ⇒ A	X <sub>L</sub> ⇒ A	Y <sub>L</sub> ⇒ A	SP <sub>L</sub> ⇒ A
1	A ⇒ B	B ⇒ B	CCR ⇒ B	TMP3 <sub>L</sub> ⇒ B	B ⇒ B	X <sub>L</sub> ⇒ B	Y <sub>L</sub> ⇒ B	SP <sub>L</sub> ⇒ B
2	A ⇒ CCR	B ⇒ CCR	CCR ⇒ CCR	TMP3 <sub>L</sub> ⇒ CCR	B ⇒ CCR	X <sub>L</sub> ⇒ CCR	Y <sub>L</sub> ⇒ CCR	SP <sub>L</sub> ⇒ CCR
3	sex:A ⇒ TMP2	sex:B ⇒ TMP2	sex:CCR ⇒ TMP2	TMP3 ⇒ TMP2	D ⇒ TMP2	X ⇒ TMP2	Y ⇒ TMP2	SP ⇒ TMP2
4	sex:A ⇒ D SEX A,D	sex:B ⇒ D SEX B,D	sex:CCR ⇒ D SEX CCR,D	TMP3 ⇒ D	D ⇒ D	X ⇒ D	Y ⇒ D	SP ⇒ D
5	sex:A ⇒ X SEX A,X	sex:B ⇒ X SEX B,X	sex:CCR ⇒ X SEX CCR,X	TMP3 ⇒ X	D ⇒ X	X ⇒ X	Y ⇒ X	SP ⇒ X
6	sex:A ⇒ Y SEX A,Y	sex:B ⇒ Y SEX B,Y	sex:CCR ⇒ Y SEX CCR,Y	TMP3 ⇒ Y	D ⇒ Y	X ⇒ Y	Y ⇒ Y	SP ⇒ Y
7	sex:A ⇒ SP SEX A,SP	sex:B ⇒ SP SEX B,SP	sex:CCR ⇒ SP SEX CCR,SP	TMP3 ⇒ SP	D ⇒ SP	X ⇒ SP	Y ⇒ SP	SP ⇒ SP

# TPA

Transfer from Condition Code  
Register to Accumulator A

# TPA

**Operation:** (CCR)  $\Rightarrow$  A

**Description:** Transfers the content of the condition code register to corresponding bit positions of accumulator A. The CCR remains unchanged.

This mnemonic is implemented by the TFR CCR,A instruction. For compatibility with the M68HC11, the mnemonic TPA is translated into the TFR CCR,A instruction by the assembler.

## Condition Codes and Boolean Formulas:

S	X	H	I	N	Z	V	C
—	—	—	—	—	—	—	—

None affected.

## Addressing Modes, Machine Code, and Execution Times:

Source Form	Address Mode	Object Code	Cycles	Access Detail
TPA <i>translates to...</i> TFR CCR,A	INH	B7 20	1	P

# TRAP

## Unimplemented Opcode Trap

# TRAP

**Operation:** (SP) - \$0002  $\Rightarrow$  SP; RTN<sub>H</sub> : RTN<sub>L</sub>  $\Rightarrow$  (M<sub>(SP)</sub> : M<sub>(SP + 1)</sub>)  
(SP) - \$0002  $\Rightarrow$  SP; Y<sub>H</sub> : Y<sub>L</sub>  $\Rightarrow$  (M<sub>(SP)</sub> : M<sub>(SP + 1)</sub>)  
(SP) - \$0002  $\Rightarrow$  SP; X<sub>H</sub> : X<sub>L</sub>  $\Rightarrow$  (M<sub>(SP)</sub> : M<sub>(SP + 1)</sub>)  
(SP) - \$0002  $\Rightarrow$  SP; B : A  $\Rightarrow$  (M<sub>(SP)</sub> : M<sub>(SP + 1)</sub>)  
(SP) - \$0001  $\Rightarrow$  SP; CCR  $\Rightarrow$  (M<sub>(SP)</sub>)  
1  $\Rightarrow$  I  
(Trap Vector)  $\Rightarrow$  PC

**Description:** Traps unimplemented opcodes. There are opcodes in all 256 positions in the page 1 opcode map, but only 54 of the 256 positions on page 2 of the opcode map are used. If the CPU attempts to execute one of the unimplemented opcodes on page 2, an opcode trap interrupt occurs. Unimplemented opcode traps are essentially interrupts that share the \$FFF8:\$FFF9 interrupt vector.

TRAP uses the next address after the unimplemented opcode as a return address. It stacks the return address, index registers Y and X, accumulators B and A, and the CCR, automatically decrementing the SP before each item is stacked. The I mask bit is then set, the PC is loaded with the trap vector, and instruction execution resumes at that location. This instruction is not maskable by the I bit. Refer to **SECTION 7 EXCEPTION PROCESSING** for more information.

### Condition Codes and Boolean Formulas:

S	X	H	I	N	Z	V	C
-	-	-	1	-	-	-	-

I: 1; Set.

### Addressing Modes, Machine Code, and Execution Times:

Source Form	Address Mode	Object Code	Cycles	Access Detail
TRAP <i>trapnum</i>	INH	\$18 tn <sup>1</sup>	11	0EVSPSSPSsP

Notes:

1. The value tn represents an unimplemented page 2 opcode in either of the two ranges \$30 to \$39 or \$40 to \$FF.

# TST

## Test Memory

# TST

**Operation:** (M) – \$00

**Description:** Subtracts \$00 from the content of memory location M and sets the condition codes accordingly.

The subtraction is accomplished internally without modifying M.

The TST instruction provides limited information when testing unsigned values. Since no unsigned value is less than zero, BLO and BLS have no utility following TST. While BHI can be used after TST, it performs the same function as BNE, which is preferred. After testing signed values, all signed branches are available.

### Condition Codes and Boolean Formulas:

S	X	H	I	N	Z	V	C
–	–	–	–	Δ	Δ	0	0

- N: Set if MSB of result is set; cleared otherwise.  
Z: Set if result is \$00; cleared otherwise.  
V: 0; Cleared.  
C: 0; Cleared.

### Addressing Modes, Machine Code, and Execution Times:

Source Form	Address Mode	Object Code	Cycles	Access Detail
TST <i>opr16a</i>	EXT	F7 hh ll	3	rOP
TST <i>opr0_xyxp</i>	IDX	E7 xb	3	rFP
TST <i>opr9,xyp</i>	IDX1	E7 xb ff	3	rPO
TST <i>opr16,xyp</i>	IDX2	E7 xb ee ff	4	frPP
TST [D, <i>xyp</i> ]	[D,IDX]	E7 xb	6	fIfFP
TST [ <i>opr16,xyp</i> ]	[IDX2]	E7 xb ee ff	6	fIPrFP

# TSTA

## Test A

# TSTA

**Operation:** (A) – \$00

**Description:** Subtracts \$00 from the content of accumulator A and sets the condition codes accordingly.

The subtraction is accomplished internally without modifying A.

The TSTA instruction provides limited information when testing unsigned values. Since no unsigned value is less than zero, BLO and BLS have no utility following TSTA. While BHI can be used after TST, it performs the same function as BNE, which is preferred. After testing signed values, all signed branches are available.

### Condition Codes and Boolean Formulas:

S	X	H	I	N	Z	V	C
-	-	-	-	Δ	Δ	0	0

- N: Set if MSB of result is set; cleared otherwise.  
Z: Set if result is \$00; cleared otherwise.  
V: 0; Cleared.  
C: 0; Cleared.

### Addressing Modes, Machine Code, and Execution Times:

Source Form	Address Mode	Object Code	Cycles	Access Detail
TSTA	INH	97	1	0

# TSTB

Test B

# TSTB

**Operation:** (B) – \$00

**Description:** Subtracts \$00 from the content of accumulator B and sets the condition codes accordingly.

The subtraction is accomplished internally without modifying B.

The TSTB instruction provides limited information when testing unsigned values. Since no unsigned value is less than zero, BLO and BLS have no utility following TSTB. While BHI can be used after TST, it performs the same function as BNE, which is preferred. After testing signed values, all signed branches are available.

**Condition Codes and Boolean Formulas:**

S	X	H	I	N	Z	V	C
–	–	–	–	Δ	Δ	0	0

- N: Set if MSB of result is set; cleared otherwise.  
Z: Set if result is \$00; cleared otherwise.  
V: 0; Cleared.  
C: 0; Cleared.

**Addressing Modes, Machine Code, and Execution Times:**

Source Form	Address Mode	Object Code	Cycles	Access Detail
TSTB	INH	D7	1	0

# TSX

Transfer from Stack Pointer  
to Index Register X

# TSX

**Operation:** (SP)  $\Rightarrow$  X

**Description:** This is an alternate mnemonic to transfer the stack pointer value to index register X. The content of the SP remains unchanged. After a TSX instruction, X points at the last value that was stored on the stack.

### Condition Codes and Boolean Formulas:

S	X	H	I	N	Z	V	C
-	-	-	-	-	-	-	-

None affected.

### Addressing Modes, Machine Code, and Execution Times:

Source Form	Address Mode	Object Code	Cycles	Access Detail
TSX translates to... TFR SP,X	INH	B7 75	1	P



# TSY

Transfer from Stack Pointer  
to Index Register Y

# TSY

**Operation:** (SP)  $\Rightarrow$  Y

**Description:** This is an alternate mnemonic to transfer the stack pointer value to index register Y. The content of the SP remains unchanged. After a TSY instruction, Y points at the last value that was stored on the stack.

### Condition Codes and Boolean Formulas:

S	X	H	I	N	Z	V	C
-	-	-	-	-	-	-	-

None affected.

### Addressing Modes, Machine Code, and Execution Times:

Source Form	Address Mode	Object Code	Cycles	Access Detail
TSY translates to... TFR SP,Y	INH	B7 76	1	P

# TXS

## Transfer from Index Register X to Stack Pointer

# TXS

**Operation:** (X)  $\Rightarrow$  SP

**Description:** This is an alternate mnemonic to transfer index register X value to the stack pointer. The content of X is unchanged.

### Condition Codes and Boolean Formulas:

S	X	H	I	N	Z	V	C
-	-	-	-	-	-	-	-

None affected.

### Addressing Modes, Machine Code, and Execution Times:

Source Form	Address Mode	Object Code	Cycles	Access Detail
TXS translates to... TFR X,SP	INH	B7 57	1	P

# TYS

## Transfer from Index Register Y to Stack Pointer

# TYS

**Operation:** (Y) ⇒ SP

**Description:** This is an alternate mnemonic to transfer index register Y value to the stack pointer. The content of Y is unchanged.

**Condition Codes and Boolean Formulas:**

S	X	H	I	N	Z	V	C
—	—	—	—	—	—	—	—

None affected.

**Addressing Modes, Machine Code, and Execution Times:**

Source Form	Address Mode	Object Code	Cycles	Access Detail
TYS <i>translates to...</i> TFR Y,SP	INH	B7 67	1	P

# WAI

## Wait for Interrupt

# WAI

**Operation:** (SP) - \$0002  $\Rightarrow$  SP; RTN<sub>H</sub> : RTN<sub>L</sub>  $\Rightarrow$  (M<sub>(SP)</sub> : M<sub>(SP + 1)</sub>)  
 (SP) - \$0002  $\Rightarrow$  SP; Y<sub>H</sub> : Y<sub>L</sub>  $\Rightarrow$  (M<sub>(SP)</sub> : M<sub>(SP + 1)</sub>)  
 (SP) - \$0002  $\Rightarrow$  SP; X<sub>H</sub> : X<sub>L</sub>  $\Rightarrow$  (M<sub>(SP)</sub> : M<sub>(SP + 1)</sub>)  
 (SP) - \$0002  $\Rightarrow$  SP; B : A  $\Rightarrow$  (M<sub>(SP)</sub> : M<sub>(SP + 1)</sub>)  
 (SP) - \$0001  $\Rightarrow$  SP; CCR  $\Rightarrow$  (M<sub>(SP)</sub>)  
 Stop CPU Clocks

**Description:** Puts the CPU into a wait state. Uses the address of the instruction following WAI as a return address. Stacks the return address, index registers Y and X, accumulators B and A, and the CCR, decrementing the SP before each item is stacked.

The CPU then enters a wait state for an integer number of bus clock cycles. During the wait state, CPU clocks are stopped, but other MCU clocks can continue to run. The CPU leaves the wait state when it senses an interrupt that has not been masked.

Upon leaving the wait state, the CPU sets the appropriate interrupt mask bit(s), fetches the vector corresponding to the interrupt sensed, and instruction execution continues at the location the vector points to.

### Condition Codes and Boolean Formulas:

S	X	H	I	N	Z	V	C
-	-	-	-	-	-	-	-

Although the WAI instruction itself does not alter the condition codes, the interrupt that causes the CPU to resume processing also causes the I mask bit (and the X mask bit, if the interrupt was  $\overline{\text{XIRQ}}$ ) to be set as the interrupt vector is fetched.

### Addressing Modes, Machine Code, and Execution Times:

Source Form	Address Mode	Object Code	Cycles	Access Detail
WAI (before interrupt)	INH	3E	8	OSSfSsf
(when interrupt comes)			5	VfPPP

# WAV

## Weighted Average

# WAV

**Operation:** Do until B = 0, leave SOP in Y : D, SOW in X

Partial Product = (M pointed to by X) × (M pointed to by Y)  
Sum-of-Products (24-bit SOP) = Previous SOP + Partial Product  
Sum-of-Weights (16-bit SOW) = Previous SOW + (M pointed to by Y)  
(X) + \$0001 ⇒ X; (Y) + \$0001 ⇒ Y  
(B) – \$01 ⇒ B

**Description:** Performs weighted average calculations on values stored in memory. Uses indexed (X) addressing mode to reference one source operand list, and indexed (Y) addressing mode to reference a second source operand list. Accumulator B is used as a counter to control the number of elements to be included in the weighted average.

For each pair of data points, a 24-bit sum of products (SOP) and a 16-bit sum of weights (SOW) is accumulated in temporary registers. When B reaches zero (no more data pairs), the SOP is placed in Y : D. The SOW is placed in X. To arrive at the final weighted average, divide the content of Y : D by X by executing an EDIV after the WAV.

This instruction can be interrupted. If an interrupt occurs during WAV execution, the intermediate results (six bytes) are stacked in the order SOW[15:0], SOP[15:0], \$00:SOP[23:16] before the interrupt is processed. The wavr pseudo-instruction is used to resume execution after an interrupt. The mechanism is re-entrant. New WAV instructions can be started and interrupted while a previous WAV instruction is interrupted.

This instruction is often used in fuzzy logic rule evaluation. Refer to **SECTION 9 FUZZY LOGIC SUPPORT** for more information.

### Condition Codes and Boolean Formulas:

S	X	H	I	N	Z	V	C
–	–	?	–	?	1	?	?

Z: 1; Set.  
H, N, V and C may be altered by this instruction.

### Addressing Modes, Machine Code, and Execution Times:

Source Form	Address Mode	Object Code	Cycles	Access Detail
WAV (add if interrupted)	Special	18 3C	See note <sup>1</sup>	Off(frrffffff)O SSSUUrr

Notes:  
1. The 8-cycle sequence in parentheses represents the loop for one iteration of SOP and SOW accumulation.

# XGDX

Exchange Double Accumulator  
and Index Register X

# XGDX

**Operation:** (D)  $\leftrightarrow$  (X)

**Description:** Exchanges the content of double accumulator D and the content of index register X. For compatibility with the M68HC11, the XGDX instruction is translated into an EXG D,X instruction by the assembler.

**Condition Codes and Boolean Formulas:**

S	X	H	I	N	Z	V	C
-	-	-	-	-	-	-	-

None affected.

**Addressing Modes, Machine Code, and Execution Times:**

Source Form	Address Mode	Object Code	Cycles	Access Detail
XGDX translates to... EXG D,X	INH	B7 C5	1	P

# XGDY

Exchange Double Accumulator  
and Index Register Y

# XGDY

**Operation:** (D)  $\Leftrightarrow$  (Y)

**Description:** Exchanges the content of double accumulator D and the content of index register Y. For compatibility with the M68HC11, the XGDY instruction is translated into an EXG D,Y instruction by the assembler.

**Condition Codes and Boolean Formulas:**

S	X	H	I	N	Z	V	C
—	—	—	—	—	—	—	—

None affected.

**Addressing Modes, Machine Code, and Execution Times:**

Source Form	Address Mode	Object Code	Cycles	Access Detail
XGDY <i>translates to...</i> EXG D,Y	INH	B7 C6	1	P

## SECTION 7 EXCEPTION PROCESSING

Exceptions are events that require processing outside the normal flow of instruction execution. This section describes exceptions and the way each is handled.

### 7.1 Types of Exceptions

CPU12 exceptions include resets, an unimplemented opcode trap, a software interrupt instruction, X-bit interrupts, and I-bit interrupts. Each exception has an associated 16-bit vector, which points to the memory location where the routine that handles the exception is located. As shown in **Table 7-1**, vectors are stored in the upper 128 bytes of the standard 64-Kbyte address map.

**Table 7-1 CPU12 Exception Vector Map**

Vector Address	Source
\$FFFE-\$FFFF	System Reset
\$FFFC-\$FFFD	Clock Monitor Reset
\$FFFA-\$FFFB	COP Reset
\$FFF8-\$FFF9	Unimplemented Opcode Trap
\$FFF6-\$FFF7	Software Interrupt Instruction (SWI)
\$FFF4-\$FFF5	XIRQ Signal
\$FFF2-\$FFF3	IRQ Signal
\$FFC0-\$FFF1	Device-Specific Interrupt Sources

The six highest vector addresses are used for resets and unmaskable interrupt sources. The remaining vectors are used for maskable interrupts. All vectors must be programmed to point to the address of the appropriate service routine.

The CPU12 can handle up to 64 exception vectors, but the number actually used varies from device to device, and some vectors are reserved for Motorola use. Refer to device documentation for more information.

Exceptions can be classified by the effect of the X and I interrupt mask bits on recognition of a pending request.

Resets, the unimplemented opcode trap, and the SWI instruction are not affected by the X and I mask bits.

Interrupt service requests from the  $\overline{\text{XIRQ}}$  pin are inhibited when  $X = 1$ , but are not affected by the I bit.

All other interrupts are inhibited when  $I = 1$ .



## 7.2 Exception Priority

A hardware priority hierarchy determines which reset or interrupt is serviced first when simultaneous requests are made. Six sources are not maskable. The remaining sources are maskable, and the device integration module typically can change the relative priorities of maskable interrupts. Refer to **7.4 Interrupts** for more detail concerning interrupt priority and servicing.

The priorities of the unmaskable sources are:

1.  $\overline{\text{RESET}}$  pin
2. Clock monitor reset
3. COP watchdog reset
4.  $\overline{\text{XIRQ}}$  signal
5. Unimplemented opcode trap
6. Software interrupt instruction (SWI)

An external reset has the highest exception-processing priority, followed by clock monitor reset, and then the on-chip watchdog reset.

The  $\overline{\text{XIRQ}}$  interrupt is pseudo-non-maskable. After reset, the X bit in the CCR is set, which inhibits all interrupt service requests from the  $\overline{\text{XIRQ}}$  pin until the X bit is cleared. The X bit can be cleared by a program instruction, but program instructions cannot reset X from zero to one. Once the X bit is cleared, interrupt service requests made via the  $\overline{\text{XIRQ}}$  pin become non-maskable.

The unimplemented page 2 opcode trap (TRAP) and the software interrupt instruction (SWI) are special cases. In one sense, these two exceptions have very low priority, because any enabled interrupt source that is pending prior to the time exception processing begins will take precedence. However, once the CPU begins processing a TRAP or SWI, neither can be interrupted. Also, since these are mutually exclusive instructions, they have no relative priority.

All remaining interrupts are subject to masking via the I bit in the CCR. Most M68HC12 MCUs have an external  $\overline{\text{IRQ}}$  pin, which is assigned the highest I-bit interrupt priority, and an internal periodic real-time interrupt generator, which has the next highest priority. The other maskable sources have default priorities that follow the address order of the interrupt vectors — the higher the address, the higher the priority of the interrupt. Other maskable interrupts are associated with on-chip peripherals such as timers or serial ports. Typically, logic in the device integration module can give one I-masked source priority over other I-masked sources. Refer to the documentation for the specific M68HC12 derivative for more information.

## 7.3 Resets

M68HC12 devices perform resets with a combination of hardware and software. Integration module circuitry determines the type of reset that has occurred, performs basic system configuration, then passes control to the CPU12. The CPU fetches a vector determined by the type of reset that has occurred, jumps to the address pointed to by the vector, and begins to execute code at that address.

There are four possible sources of reset. Power-on reset (POR) and external reset share the same reset vector. The computer operating properly (COP) reset and the clock monitor reset each have a vector.

### 7.3.1 Power-On Reset

The M68HC12 device integration module incorporates circuitry to detect a positive transition in the  $V_{DD}$  supply and initialize the device during cold starts, generally by asserting the reset signal internally. The signal is typically released after a delay that allows the device clock generator to stabilize.

### 7.3.2 External Reset

The MCU distinguishes between internal and external resets by sensing how quickly the signal on the RESET pin rises to logic level one after it has been asserted. When the MCU senses any of the four reset conditions, internal circuitry drives the RESET signal low for 16 clock cycles, then releases. Eight clock cycles later, the MCU samples the state of the signal applied to the RESET pin. If the signal is still low, an external reset has occurred. If the signal is high, reset has been initiated internally by either the COP system or the clock monitor.

### 7.3.3 COP Reset

The MCU includes a computer operating properly (COP) system to help protect against software failures. When the COP is enabled, software must write a particular code sequence to a specific address in order to keep a watchdog timer from timing out. If software fails to execute the sequence properly, a reset occurs.

### 7.3.4 Clock Monitor Reset

The clock monitor circuit uses an internal RC circuit to determine whether clock frequency is above a predetermined limit. If clock frequency falls below the limit when the clock monitor is enabled, a reset occurs.

## 7.4 Interrupts

Each M68HC12 device can recognize a number of interrupt sources. Each source has a vector in the vector table. The XIRQ signal, the unimplemented opcode trap, and the SWI instruction are non-maskable, and have a fixed priority. The remaining interrupt sources can be masked by the I bit. In most M68HC12 devices, the external interrupt request pin is assigned the highest maskable interrupt priority, and the internal periodic real-time interrupt generator has the next highest priority. Other maskable interrupts are associated with on-chip peripherals such as timers or serial ports. These maskable sources have default priorities that follow the address order of the interrupt vectors. The higher the vector address, the higher the priority of the interrupt. Typically, a device integration module incorporates logic that can give one maskable source priority over other maskable sources.

#### 7.4.1 Non-Maskable Interrupt Request ( $\overline{\text{XIRQ}}$ )

The  $\overline{\text{XIRQ}}$  input is an updated version of the  $\overline{\text{NMI}}$  input of earlier MCUs. The  $\overline{\text{XIRQ}}$  function is disabled during system reset and upon entering the interrupt service routine for an  $\overline{\text{XIRQ}}$  interrupt.

During reset, both the I bit and the X bit in the CCR are set. This disables maskable interrupts and interrupt service requests made by asserting the  $\overline{\text{XIRQ}}$  signal. After minimum system initialization, software can clear the X bit using an instruction such as `ANDCC #$BF`. Software cannot reset the X bit from zero to one once it has been cleared, and interrupt requests made via the  $\overline{\text{XIRQ}}$  pin become non-maskable. When a non-maskable interrupt is recognized, both the X and I bits are set after context is saved. The X bit is not affected by maskable interrupts. Execution of an RTI at the end of the interrupt service routine normally restores the X and I bits to the pre-interrupt request state.

#### 7.4.2 Maskable Interrupts

Maskable interrupt sources include on-chip peripheral systems and external interrupt service requests. Interrupts from these sources are recognized when the global interrupt mask bit (I) in the CCR is cleared. The default state of the I bit out of reset is one, but it can be written at any time.

The integration module manages maskable interrupt priorities. Typically, an on-chip interrupt source is subject to masking by associated bits in control registers in addition to global masking by the I bit in the CCR. Sources generally must be enabled by writing one or more bits in associated control registers. There may be other interrupt-related control bits and flags, and there may be specific register read-write sequences associated with interrupt service. Refer to individual on-chip peripheral descriptions for details.

#### 7.4.3 Interrupt Recognition

Once enabled, an interrupt request can be recognized at any time after the I mask bit is cleared. When an interrupt service request is recognized, the CPU responds at the completion of the instruction being executed. Interrupt latency varies according to the number of cycles required to complete the current instruction. Because the REV, REVW and WAV instructions can take many cycles to complete, they are designed so that they can be interrupted. Instruction execution resumes when interrupt execution is complete. When the CPU begins to service an interrupt, the instruction queue is re-filled, a return address is calculated, and then the return address and the contents of the CPU registers are stacked as shown in **Table 7-2**.

After the CCR is stacked, the I bit (and the X bit, if an  $\overline{\text{XIRQ}}$  interrupt service request caused the interrupt) is set to prevent other interrupts from disrupting the interrupt service routine. Execution continues at the address pointed to by the vector for the highest-priority interrupt that was pending at the beginning of the interrupt sequence. At the end of the interrupt service routine, an RTI instruction restores context from the stacked registers, and normal program execution resumes.

**Table 7-2 Stacking Order on Entry to Interrupts**

Memory Location	CPU Registers
SP - 2	RTN <sub>H</sub> : RTN <sub>L</sub>
SP - 4	Y <sub>H</sub> : Y <sub>L</sub>
SP - 6	X <sub>H</sub> : X <sub>L</sub>
SP - 8	B : A
SP - 9	CCR

#### 7.4.4 External Interrupts

External interrupt service requests are made by asserting an active-low signal connected to the  $\overline{\text{IRQ}}$  pin. Typically, control bits in the device integration module affect how the signal is detected and recognized.

The I bit serves as the  $\overline{\text{IRQ}}$  interrupt enable flag. When an  $\overline{\text{IRQ}}$  interrupt is recognized, the I bit is set to inhibit interrupts during the interrupt service routine. Before other maskable interrupt requests can be recognized, the I bit must be cleared. This is generally done by an RTI instruction at the end of the service routine.

#### 7.4.5 Return from Interrupt Instruction (RTI)

RTI is used to terminate interrupt service routines. RTI is an 8-cycle instruction when no other interrupt is pending, and a 10-cycle instruction when another interrupt is pending. In either case, the first five cycles are used to restore (pull) the CCR, B:A, X, Y, and the return address from the stack. If no other interrupt is pending at this point, three program words are fetched to refill the instruction queue from the area of the return address and processing proceeds from there.

If another interrupt is pending after registers are restored, a new vector is fetched, and the stack pointer is adjusted to point at the CCR value that was just recovered (SP = SP - 9). This makes it appear that the registers have been stacked again. After the SP is adjusted, three program words are fetched to refill the instruction queue, starting at the address the vector points to. Processing then continues with execution of the instruction that is now at the head of the queue.

#### 7.5 Unimplemented Opcode Trap

The CPU12 has opcodes in all 256 positions in the page 1 opcode map, but only 54 of the 256 positions on page 2 of the opcode map are used. If the CPU attempts to execute one of the 202 unused opcodes on page 2, an unimplemented opcode trap occurs. The 202 unimplemented opcodes are essentially interrupts that share a common interrupt vector, \$FFF8:\$FFF9.

The CPU12 uses the next address after an unimplemented page 2 opcode as a return address. This differs from the M68HC11 illegal opcode interrupt, which uses the address of an illegal opcode as the return address. In the CPU12, the stacked return address can be used to calculate the address of the unimplemented opcode for software-controlled traps.

## 7.6 Software Interrupt Instruction

Execution of the SWI instruction causes an interrupt without an interrupt service request. SWI is not inhibited by the global mask bits in the CCR, and execution of SWI sets the I mask bit. Once an SWI interrupt begins, maskable interrupts are inhibited until the I bit in the CCR is cleared. This typically occurs when an RTI instruction at the end of the SWI service routine restores context.

## 7.7 Exception Processing Flow

The first cycle in the exception processing flow for all CPU12 exceptions is the same, regardless of the source of the exception. Between the first and second cycles of execution, the CPU chooses one of three alternative paths. The first path is for resets, the second path is for pending X or I interrupts, and the third path is used for software interrupts (SWI) and trapping unimplemented opcodes. The last two paths are virtually identical, differing only in the details of calculating the return address. Refer to **Figure 7-2** for the following discussion.

### 7.7.1 Vector Fetch

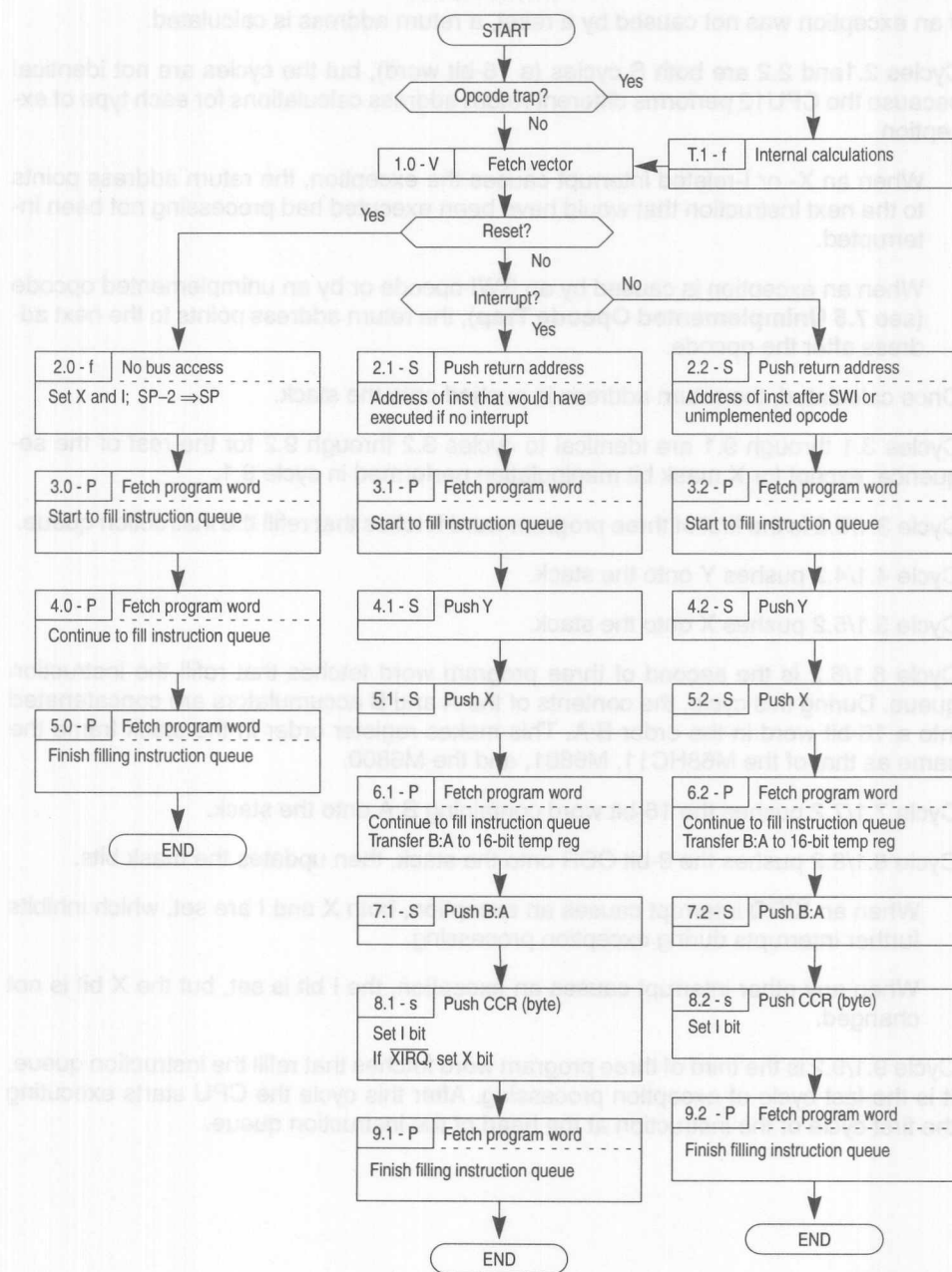
The first cycle of all exception processing, regardless of the cause, is a vector fetch. The vector points to the address where exception processing will continue. Exception vectors are stored in a table located at the top of the memory map (\$FFC0–\$FFFF). The CPU cannot use the fetched vector until the third cycle of the exception processing sequence.

During the vector fetch cycle, the CPU issues a signal that tells the integration module to drive the vector address of the highest priority, pending exception onto the system address bus (the CPU does not provide this address).

After the vector fetch, the CPU selects one of the three alternate execution paths, depending upon the cause of the exception.

### 7.7.2 Reset Exception Processing

If reset caused the exception, processing continues to cycle 2.0. This cycle sets the X and I bits in the CCR. The stack pointer is also decremented by two, but this is an artifact of shared code used for interrupt processing; the SP is not intended to have any specific value after a reset. Cycles 3.0 through 5.0 are program word fetches that refill the instruction queue. Fetches start at the address pointed to by the reset vector. When the fetches are completed, exception processing ends, and the CPU starts executing the instruction at the head of the instruction queue.



CPU12EXPFLOW

Figure 7-2 Exception Processing Flow Diagram

### 7.7.3 Interrupt and Unimplemented Opcode Trap Exception Processing

If an exception was not caused by a reset, a return address is calculated.

Cycles 2.1 and 2.2 are both S cycles (a 16-bit word), but the cycles are not identical because the CPU12 performs different return address calculations for each type of exception.

When an X- or I-related interrupt causes the exception, the return address points to the next instruction that would have been executed had processing not been interrupted.

When an exception is caused by an SWI opcode or by an unimplemented opcode (see **7.5 Unimplemented Opcode Trap**), the return address points to the next address after the opcode.

Once calculated, the return address is pushed onto the stack.

Cycles 3.1 through 9.1 are identical to cycles 3.2 through 9.2 for the rest of the sequence, except for X mask bit manipulation performed in cycle 8.1.

Cycle 3.1/3.2 is the first of three program word fetches that refill the instruction queue.

Cycle 4.1/4.2 pushes Y onto the stack.

Cycle 5.1/5.2 pushes X onto the stack.

Cycle 6.1/6.2 is the second of three program word fetches that refill the instruction queue. During this cycle, the contents of the A and B accumulators are concatenated into a 16-bit word in the order B:A. This makes register order in the stack frame the same as that of the M68HC11, M6801, and the M6800.

Cycle 7.1/7.2 pushes the 16-bit word containing B:A onto the stack.

Cycle 8.1/8.2 pushes the 8-bit CCR onto the stack, then updates the mask bits.

When an  $\overline{\text{XIRQ}}$  interrupt causes an exception, both X and I are set, which inhibits further interrupts during exception processing.

When any other interrupt causes an exception, the I bit is set, but the X bit is not changed.

Cycle 9.1/9.2 is the third of three program word fetches that refill the instruction queue. It is the last cycle of exception processing. After this cycle the CPU starts executing the first cycle of the instruction at the head of the instruction queue.



## SECTION 8

### DEVELOPMENT AND DEBUG SUPPORT

This section is an explanation of CPU-related aspects of the background debugging system. Topics include the instruction queue status signals, instruction tagging, and the single-wire background debug interface.

#### 8.1 External Reconstruction of the Queue

The CPU12 uses an instruction queue to buffer program information and increase instruction throughput. The queue consists of two 16-bit stages, plus a 16-bit holding latch. Program information is always fetched in aligned 16-bit words. At least three bytes of program information are available to the CPU when instruction execution begins. The holding latch is used when a word of program information arrives before the queue can advance.

Because of the queue, program information is fetched a few cycles before it is used by the CPU. Internally, the MCU only needs to buffer the fetched data. But, in order to monitor cycle-by-cycle CPU activity, it is necessary to externally reconstruct what is happening in the instruction queue.

Two external pins, IPIPE[1:0], provide time-multiplexed information about data movement in the queue and instruction execution. To complete the picture for system debugging, it is also necessary to include program information and associated addresses in the reconstructed queue.

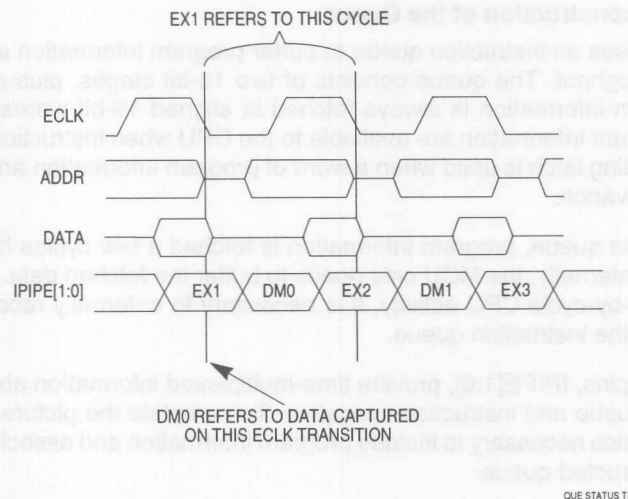
The instruction queue and cycle-by-cycle activity can be reconstructed in real time or from trace history captured by a logic analyzer. However, neither scheme can be used to stop the CPU12 at a specific instruction. By the time an operation is visible outside the MCU, the instruction has already begun execution. A separate instruction tagging mechanism is provided for this purpose. A tag follows the information in the queue as the queue is advanced. During debugging, the CPU enters active background debug mode when a tagged instruction reaches the head of the queue, rather than executing the tagged instruction. For more information about tagging, refer to **8.5 Instruction Tagging**.

#### 8.2 Instruction Queue Status Signals

The IPIPE[1:0] signals carry time-multiplexed information about data movement and instruction execution during normal CPU operation. The signals are available on two multifunctional device pins. During reset, the pins are used as mode-select input signals MODA and MODB. After reset, information on the pins does not become valid until an instruction reaches queue stage 2.



To reconstruct the queue, the information carried by the status signals must be captured externally. In general, data movement and execution start information are considered to be distinct 2-bit values, with the low-order bit on IPIPE0 and the high-order bit on IPIPE1. Data movement information is available on rising edges of the E clock; execution start information is available on falling edges of the E clock, as shown in **Figure 8-1**. Data movement information refers to data on the bus at the previous falling edge of E. Execution information refers to the bus cycle from the current falling edge of E to the next falling edge of E. **Table 8-1** summarizes the information encoded on the IPIPE[1:0] pins.



**Figure 8-1 Queue Status Signal Timing**

**Table 8-1 IPIPE[1:0] Decoding**

Data Movement (capture at E rise)	Mnemonic	Meaning
0:0	—	No movement
0:1	LAT	Latch data from bus
1:0	ALD	Advance queue and load from bus
1:1	ALL	Advance queue and load from latch
Execution Start (capture at E fall)	Mnemonic	Meaning
0:0	—	No start
0:1	INT	Start interrupt sequence
1:0	SEV	Start even instruction
1:1	SOD	Start odd instruction

### 8.2.1 Zero Encoding (0:0)

The 0:0 state at the rising edge of E indicates that there was no data movement in the instruction queue during the previous cycle; the 0:0 state at the falling edge of E indicates continuation of an instruction or interrupt sequence.

### 8.2.2 LAT — Latch Data from Bus Encoding (0:1)

Fetches program information has arrived, but the queue is not ready to advance. The information is latched into the buffer. Later, when the queue does advance, stage 1 is refilled from the buffer, or from the data bus if the buffer is empty. In some instruction sequences, there can be several latch cycles before the queue advances. In these cases, the buffer is filled on the first latch event and additional latch requests are ignored.

### 8.2.3 ALD — Advance and Load from Data Bus Encoding (1:0)

The two-stage instruction queue is advanced by one word and stage 1 is refilled with a word of program information from the data bus. The CPU requested the information two bus cycles earlier but, due to access delays, the information was not available until the E cycle immediately prior to the ALD.

### 8.2.4 ALL — Advance and Load from Latch Encoding (1:1)

The two-stage instruction queue is advanced by one word and stage 1 is refilled with a word of program information from the buffer. The information was latched from the data bus at the falling edge of a previous E cycle because the instruction queue was not ready to advance when it arrived.

### 8.2.5 INT — Interrupt Sequence Encoding (0:1)

The E cycle starting at this E fall is the first cycle of an interrupt sequence. Normally this cycle is a read of the interrupt vector. However, in systems that have interrupt vectors in external memory and an 8-bit data bus, this cycle reads only the upper byte of the 16-bit interrupt vector.

### 8.2.6 SEV — Start Instruction on Even Address Encoding (1:0)

The E cycle starting at this E fall is the first cycle of the instruction in the even (high order) half of the word at the head of the instruction queue. The queue treats the \$18 prebyte for instructions on page 2 of the opcode map as a special 1-byte, 1-cycle instruction, except that interrupts are not recognized at the boundary between the prebyte and the rest of the instruction.

### 8.2.7 SOD — Start Instruction on Odd Address Encoding (1:1)

The E cycle starting at this E fall is the first cycle of the instruction in the odd (low order) half of the word at the head of the instruction queue. The queue treats the \$18 prebyte for instructions on page 2 of the opcode map as a special 1-byte, 1-cycle instruction, except that interrupts are not recognized at the boundary between the prebyte and the rest of the instruction.

### 8.3 Implementing Queue Reconstruction

The raw signals required for queue reconstruction are the address bus (ADDR), the data bus (DATA), the read/write strobe (R/W), the system clock (E), and the queue status signals (IPIPE[1:0]). An E clock cycle begins after an E fall. Addresses, R/W state, and data movement status must be captured at the E rise in the middle of the cycle. Data and execution start status must be captured at the E fall at the end of the cycle. These captures can then be organized into records with one record per E clock cycle.

Implementation details depend upon the type of device and the mode of operation. For instance, the data bus can be eight bits or 16 bits wide, and non-multiplexed or multiplexed. In all cases, the externally reconstructed queue must use 16-bit words. Demultiplexing and assembly of 8-bit data into 16-bit words is done before program information enters the real queue, so it must also be done for the external reconstruction. An example:

Systems with an 8-bit data bus and a program stored in external memory require two cycles for each program word fetch. MCU bus control logic freezes the CPU clocks long enough to do two 8-bit accesses rather than a single 16-bit access, so the CPU sees only 16-bit words of program information. To recover the 16-bit program words externally, latch the data bus state at the falling edge of E when ADDR0 = 0, and gate the outputs of the latch onto DATA[15:8] when a LAT or ALD cycle occurs. Since the 8-bit data bus is connected to DATA[7:0], the 16-bit word on the data lines corresponds to the ALD or LAT status indication at the E rise after the second 8-bit fetch, which is always to an odd address. IPIPE[1:0] status signals indicate 0:0 at the beginning (E fall) and middle (E rise) of the first 8-bit fetch.

Some M68HC12 devices have address lines to support memory expansion beyond the standard 64-Kbyte address space. When memory expansion is used, expanded addresses must also be captured and maintained.

#### 8.3.1 Queue Status Registers

Queue reconstruction requires the following registers, which can be implemented as software variables when previously captured trace data is used, or as hardware latches in real time.

##### 8.3.1.1 in\_add, in\_dat Registers

These registers contain the address and data from the previous external bus cycle. Depending upon how records are read and processed from the raw capture information, it may be possible to simply read this information from the raw capture data file when needed.

##### 8.3.1.2 fetch\_add, fetch\_dat Registers

These registers buffer address and data for information that was fetched before the queue was ready to advance.

### 8.3.1.3 st1\_add, st1\_dat Registers

These registers contain address and data for the first stage of the reconstructed instruction queue.

### 8.3.1.4 st2\_add, st2\_dat Registers

These registers contain address and data for the final stage of the reconstructed instruction queue. When the IPIPE[1:0] signals indicate that an instruction is starting to execute, the address and opcode can be found in these registers.

## 8.3.2 Reconstruction Algorithm

This section describes in detail how to use IPIPE[1:0] signals and status storage registers to perform queue reconstruction. An "is\_full" flag is used to indicate when the fetch\_add and fetch\_dat buffer registers contain information. The use of the flag is explained more fully in subsequent paragraphs.

Typically, the first few cycles of raw capture data are not useful because it takes several cycles before an instruction propagates to the head of the queue. During these first raw cycles, the only meaningful information available are data movement signals. Information on the external address and data buses during this setup time reflects the actions of instructions that were fetched before data collection started.

In the special case of a reset, there is a five cycle sequence (VfPPP) during which the reset vector is fetched and the instruction queue is filled, before execution of the first instruction begins. Due to the timing of the switchover of the IPIPE[1:0] pins from their alternate function as mode select inputs, the status information on these two pins may be erroneous during the first cycle or two after the release of reset. This is not a problem because the status is correct in time for queue reconstruction logic to correctly replicate the queue.

Before starting to reconstruct the queue, clear the is\_full flag to indicate that there is no meaningful information in the fetch\_add and fetch\_dat buffers. Further movement of information in the instruction queue is based on the decoded status on the IPIPE[1:0] signals at the rising edges of E.

### 8.3.2.1 LAT Decoding

On a latch cycle, check the is\_full flag. If and only if is\_full = 0, transfer the address and data from the previous bus cycle (in\_add and in\_dat) into the fetch\_add and fetch\_dat registers respectively. Then, set the is\_full flag. The usual reason for a latch request instead of an advance request is that the previous instruction ended with a single aligned byte of program information in the last stage of the instruction queue. Since the odd half of this word still holds the opcode for the next instruction, the queue cannot advance on this cycle. However, the cycle to fetch the next word of program information has already started and the data is on its way.

### 8.3.2.2 ALD Decoding

On an advance-and-load-from-data-bus cycle, the information in the instruction queue must advance by one stage. Whatever was in stage 2 of the queue is simply thrown away. The previous contents of stage 1 are moved to stage 2, and the address and data from the previous cycle (in\_add and in\_dat) are transferred into stage 1 of the instruction queue. Finally, clear the is\_full flag to indicate the buffer latch is ready for new data. Usually, there would be no useful information in the fetch buffer when an ALD cycle was encountered, but in the case of a change-of-flow, any data that was there needs to be flushed out (by clearing the is\_full flag).

### 8.3.2.3 ALL Decoding

On an advance-and-load-from-latch cycle, the information in the instruction queue must advance by one stage. Whatever was in stage 2 of the queue is simply thrown away. The previous contents of stage 1 are moved to stage 2, and the contents of the fetch buffer latch are transferred into stage 1 of the instruction queue. One or more cycles preceding the ALL cycle will have been a LAT cycle. After updating the instruction queue, clear the is\_full flag to indicate the fetch buffer is ready for new information.

## 8.4 Background Debug Mode

M68HC12 MCUs include a resident debugging system. This system is implemented with on-chip hardware rather than external software, and provides a full set of debugging options. The debugging system is less intrusive than systems used on other microcontrollers, because the control logic resides in the on-chip integration module, rather than in the CPU. Some activities, such as reading and writing memory locations, can be performed while the CPU is executing normal code with no effect on real-time system activity.

The integration module generally uses CPU dead cycles to execute debugging commands while the CPU is operating normally, but can steal cycles from the CPU when necessary. Other commands are firmware based, and require that the CPU be in active background debug mode (BDM) for execution. While BDM is active, the CPU executes a monitor program located in a small on-chip ROM.

Debugging control logic communicates with external devices serially, via the BKGD pin. This single-wire approach helps to minimize the number of pins needed for development support.

Background debug does not operate in STOP mode.

### 8.4.1 Enabling BDM

The debugger must be enabled before it can be activated. Enabling has two phases. First, the BDM ROM must be enabled by writing the ENBDM bit in the BDM status register, using a debugging command sent via the single wire interface. Once the ROM is enabled, it remains available until the next system reset, or until ENBDM is cleared by another debugging command. Second, BDM must be activated to map the ROM and BDM control registers to addresses \$FF00 to \$FFFF and put the MCU in background mode.

After the firmware is enabled, BDM can be activated by the hardware BACKGROUND command, by breakpoints tagged via the LIM breakpoint logic or the BDM tagging mechanism, and by the BGND instruction. An attempt to activate BDM before firmware has been enabled causes the MCU to resume normal instruction execution after a brief delay.

BDM becomes active at the next instruction boundary following execution of the BDM BACKGROUND command. Breakpoints can be configured to activate BDM before a tagged instruction is executed.

While BDM is active, BDM control registers are mapped to addresses \$FF00 to \$FF06. These registers are only accessible through BDM firmware or BDM hardware commands. **8.4.4 BDM Registers** describes the registers.

Some M68HC12 on-chip peripherals have a BDM control bit, which determines whether the peripheral function is available during BDM. If no bit is shown, the peripheral is active in BDM.

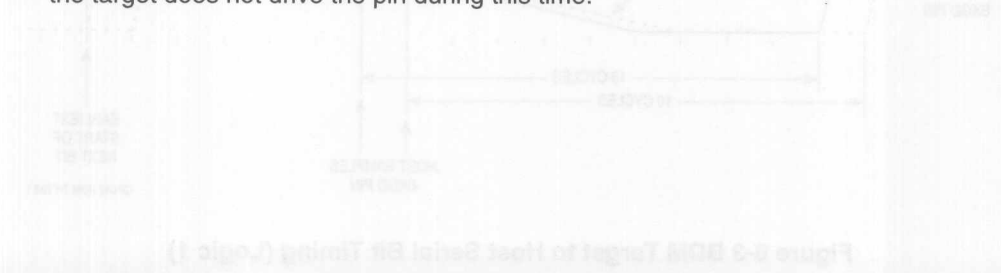
#### 8.4.2 BDM Serial Interface

The BDM serial interface uses a clocking scheme in which the external host generates a falling edge on the BKGD pin to indicate the start of each bit time. This falling edge must be sent for every bit, whether data is transmitted or received.

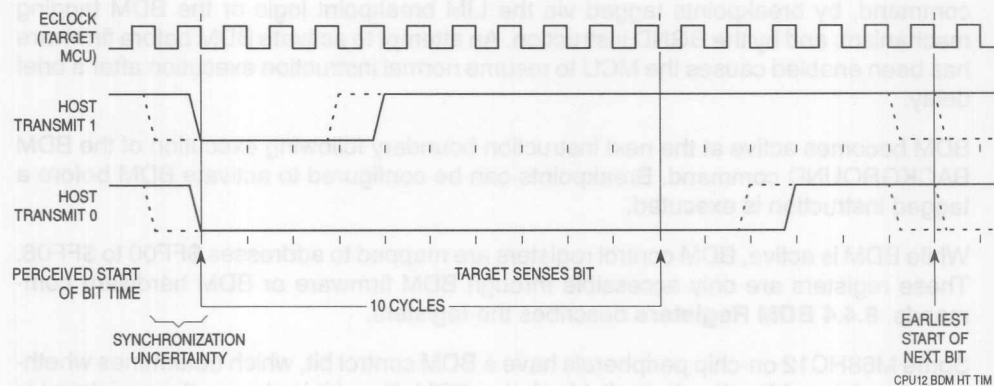
BKGD is an open drain pin that can be driven either by the MCU or by an external host. Data is transferred MSB first, at 16 E-clock cycles per bit. The interface times out if 512 E-clock cycles occur between falling edges from the host. The hardware clears the command register when a time-out occurs.

The BKGD pin is used to send and receive data. The following diagrams show timing for each of these cases. Interface timing is synchronous to MCU clocks, but the external host is asynchronous to the target MCU. The internal clock signal is shown for reference in counting cycles.

**Figure 8-2** shows an external host transmitting a data bit to the BKGD pin of a target M68HC12 MCU. The host is asynchronous to the target, so there is a 0- to 1-cycle delay from the host-generated falling edge to the time when the target perceives the bit. Ten target E-cycles later, the target senses the bit level on the BKGD pin. The host can drive high during host-to-target transmission to speed up rising edges, because the target does not drive the pin during this time.

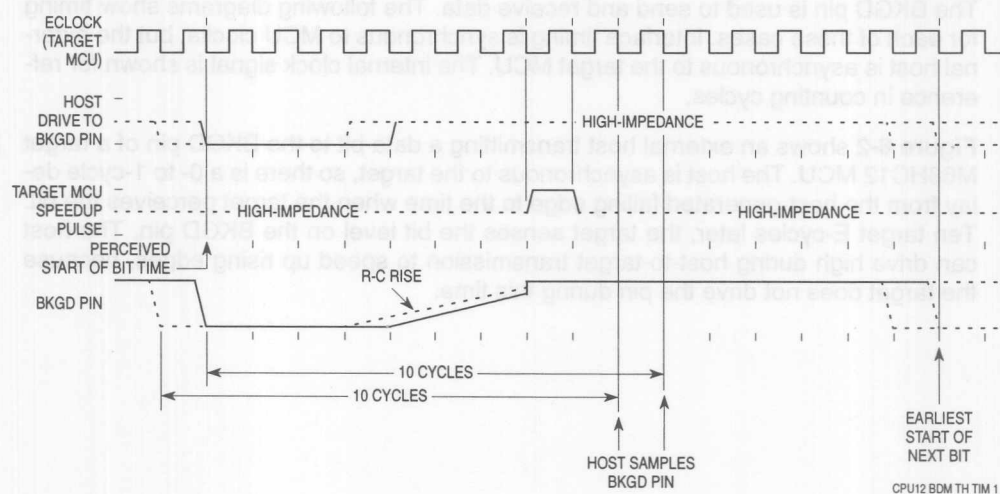






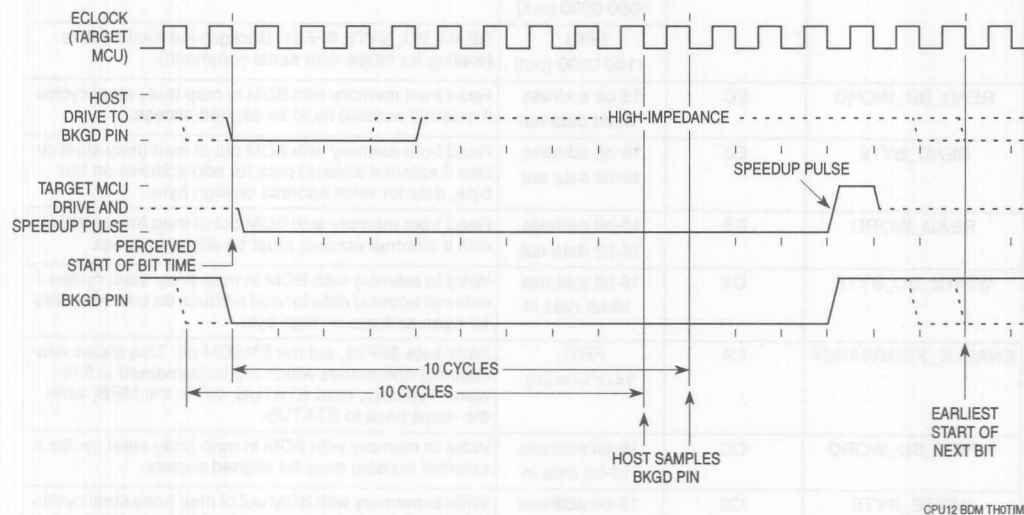
**Figure 8-2 BDM Host to Target Serial Bit Timing**

**Figure 8-3** shows an external host receiving a logic one from the target MCU. Since the host is asynchronous to the target, there is a 0- or 1-cycle delay from the host-generated falling edge on BKGD until the target perceives the bit. The host holds the signal low long enough for the target to recognize it (a minimum of two target E-clock cycles), but must release the low drive before the target begins to drive the active-high speed-up pulse seven cycles after the start of the bit time. The host should sample the bit level about ten cycles after the start of the bit time.



**Figure 8-3 BDM Target to Host Serial Bit Timing (Logic 1)**

**Figure 8-4** shows the host receiving a logic zero from the target. Since the host is asynchronous to the target, there is a 0- or 1-cycle delay from the host-generated falling edge on BKGD until the target perceives the bit. The host initiates the bit time, but the target finishes it. To make certain the host receives a logic zero, the target drives the BKGD pin low for 13 E-clock cycles, then briefly drives the signal high to speed up the rising edge. The host samples the bit level about ten cycles after starting the bit time.



**Figure 8-4 BDM Target to Host Serial Bit Timing (Logic 0)**

### 8.4.3 BDM Commands

All BDM opcodes are eight bits long, and can be followed by an address or data, as indicated by the instruction.

Commands implemented in BDM control hardware are listed in **Table 8-2**. These commands, except for BACKGROUND, do not require the CPU to be in BDM mode for execution. The control logic uses CPU dead cycles to execute these instructions. If a dead cycle cannot be found within 128 cycles, the control logic steals cycles from the CPU.



**Table 8-2 BDM Commands Implemented in Hardware**

Command	Opcode (Hex)	Data	Description
BACKGROUND	90	None	Enter background mode (if firmware enabled).
READ_BD_BYTE	E4	16-bit address 16-bit data out	Read from memory with BDM in map (may steal cycles if external access) data for odd address on low byte, data for even address on high byte.
STATUS <sup>1</sup>	E4	FF01, 0000 0000 (out)	READ_BD_BYTE \$FF01. Running user code (BGND instruction is not allowed).
		FF01, 1000 0000 (out)	READ_BD_BYTE \$FF01. BGND instruction is allowed.
		FF01, 1100 0000 (out)	READ_BD_BYTE \$FF01. Background mode active (waiting for single wire serial command).
READ_BD_WORD	EC	16-bit address 16-bit data out	Read from memory with BDM in map (may steal cycles if external access) must be aligned access.
READ_BYTE	E0	16-bit address 16-bit data out	Read from memory with BDM out of map (may steal cycles if external access) data for odd address on low byte, data for even address on high byte.
READ_WORD	E8	16-bit address 16-bit data out	Read from memory with BDM out of map (may steal cycles if external access) must be aligned access.
WRITE_BD_BYTE	C4	16-bit address 16-bit data in	Write to memory with BDM in map (may steal cycles if external access) data for odd address on low byte, data for even address on high byte.
ENABLE_FIRMWARE <sup>2</sup>	C4	FF01, 1xxx xxxx(in)	Write byte \$FF01, set the ENBDM bit. This allows execution of commands which are implemented in firmware. Typically, read STATUS, OR in the MSB, write the result back to STATUS.
WRITE_BD_WORD	CC	16-bit address 16-bit data in	Write to memory with BDM in map (may steal cycles if external access) must be aligned access.
WRITE_BYTE	C0	16-bit address 16-bit data in	Write to memory with BDM out of map (may steal cycles if external access) data for odd address on low byte, data for even address on high byte.
WRITE_WORD	C8	16-bit address 16-bit data in	Write to memory with BDM out of map (may steal cycles if external access) must be aligned access.

**NOTES:**

1. STATUS command is a specific case of the READ\_BD\_BYTE command.
2. ENABLE\_FIRMWARE is a specific case of the WRITE\_BD\_BYTE command.

The CPU must be in background mode to execute commands that are implemented in the BDM ROM. The CPU executes code from the ROM to perform the requested operation. These commands are shown in **Table 8-3**.

The host controller must wait 150 cycles for a non-intrusive BDM command to execute before another command can be sent. This delay includes 128 cycles for the maximum delay for a dead cycle.

BDM logic retains control of the internal buses until a read or write is completed. If an operation can be completed in a single cycle, it does not intrude on normal CPU operation. However, if an operation requires multiple cycles, CPU clocks are frozen until the operation is complete.

**Table 8-3 BDM Firmware Commands**

Command	Opcode (Hex)	Data	Description
GO	08	none	Resume normal processing
TRACE1	10	none	Execute one user instruction then return to BDM
TAGGO	18	none	Enable tagging then resume normal processing
WRITE_NEXT	42	16-bit data in	$X = X + 2$ ; Write next word @ 0,X
WRITE_PC	43	16-bit data in	Write program counter
WRITE_D	44	16-bit data in	Write D accumulator
WRITE_X	45	16-bit data in	Write X index register
WRITE_Y	46	16-bit data in	Write Y index register
WRITE_SP	47	16-bit data in	Write stack pointer
READ_NEXT	62	16-bit data out	$X = X + 2$ ; Read next word @ 0,X
READ_PC	63	16-bit data out	Read program counter
READ_D	64	16-bit data out	Read D accumulator
READ_X	65	16-bit data out	Read X index register
READ_Y	66	16-bit data out	Read Y index register
READ_SP	67	16-bit data out	Read stack pointer

#### 8.4.4 BDM Registers

Seven BDM registers are mapped into the standard 64-Kbyte address space when BDM is active. Mapping is shown in **Table 8-4**.

**Table 8-4 BDM Register Mapping**

Address	Register
\$FF00	BDM instruction register
\$FF01	BDM status register
\$FF02-\$FF03	BDM shift register
\$FF04-\$FF05	BDM address register
\$FF06	BDM CCR register

The content of the instruction register is determined by the type of background instruction being executed. The status register indicates BDM operating conditions. The shift register contains data being received or transmitted via the serial interface. The address register is temporary storage for BDM commands. The CCR register preserves the content of the CPU12 CCR while BDM is active.

The only register of interest to users is the status register. The other BDM registers are used only by the BDM firmware to execute commands. The registers can be accessed by means of the hardware READ\_BD and WRITE\_BD commands, but must not be written during BDM operation.

### 8.4.4.1 BDM Status Register

#### STATUS — BDM Status Register

\$FF01

	BIT 7	6	5	4	3	2	1	BIT 0
	ENBDM	BDMACT	ENTAG	SDV	TRACE	0	0	0
RESET:	0	0	0	0	0	0	0	0
SP. S. CHIP & PERIPH.:	1	0	0	0	0	0	0	0

#### ENBDM — Enable BDM ROM

Shows whether the BDM ROM is enabled. Cleared by reset.

0 = BDM ROM not enabled

1 = BDM ROM enabled, but not in memory map unless BDM is active

#### BDMACT — BDM Active Flag

Shows whether the BDM ROM is in the memory map. Cleared by reset.

0 = ROM not in map

1 = ROM in map (MCU is in active BDM)

#### ENTAG — Instruction Tagging Enable

Shows whether instruction tagging is enabled. Set by the TAGGO instruction and cleared when BDM is entered. Cleared by reset.

#### NOTE

Execute a TAGGO command to enable instruction tagging. Do not write ENTAG directly.

0 = Tagging not enabled, or BDM active

1 = Tagging active

#### SDV — Shifter Data Valid

Shows that valid data is in the serial interface shift register.

#### NOTE

SDV is used by firmware-based instructions. Do not attempt to write SDV directly.

0 = No valid data

1 = Valid Data

#### TRACE — Trace Flag

Shows when tracing is enabled.

#### NOTE

Execute a TRACE1 command to enable instruction tagging. Do not attempt to write TRACE directly.

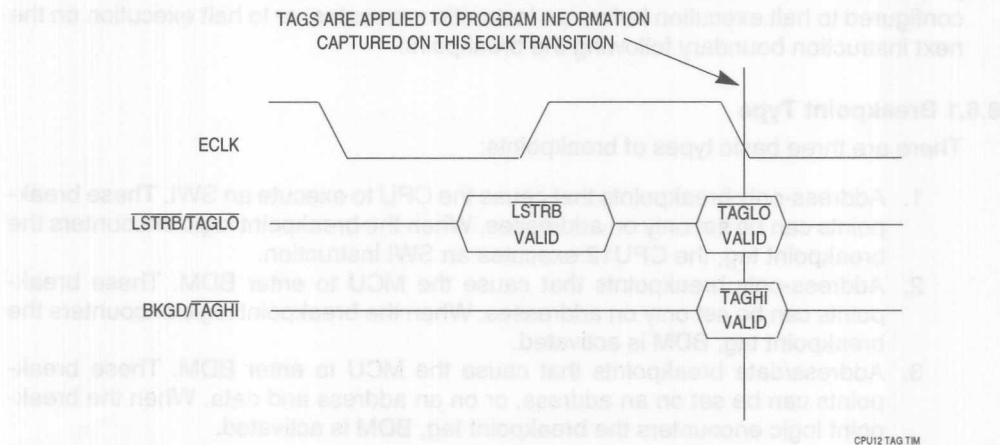
0 = Tracing not enabled

1 = Tracing active

## 8.5 Instruction Tagging

The instruction queue and cycle-by-cycle CPU activity can be reconstructed in real time, or from trace history that was captured by a logic analyzer. However, the reconstructed queue cannot be used to stop the CPU at a specific instruction, because execution has already begun by the time an operation is visible outside the MCU. A separate instruction tagging mechanism is provided for this purpose.

Executing the BDM TAGGO command configures two MCU pins for tagging. The TAGLO signal shares a pin with the LSTRB signal, and the TAGHI signal shares a pin with the BKGD pin. Tagging information is latched on the falling edge of ECLK, as shown in **Figure 8-5**.



**Figure 8-5 Tag Input Timing**

**Table 8-5** shows the functions of the two tagging pins. The pins operate independently; the state of one pin does not affect the function of the other. The presence of logic level zero on either pin at the fall of ECLK performs the indicated function. Tagging is allowed in all modes. Tagging is disabled when BDM becomes active.

**Table 8-5 Tag Pin Function**

TAGHI	TAGLO	Tag
1	1	No Tag
1	0	Low Byte
0	1	High Byte
0	0	Both Bytes

In M68HC12 derivatives that have hardware breakpoint capability, the breakpoint control logic and BDM control logic use the same internal signals for instruction tagging. The CPU12 does not differentiate between the two kinds of tags.

The tag follows program information as it advances through the queue. When a tagged instruction reaches the head of the queue, the CPU enters active background debug mode rather than executing the instruction.

## **8.6 Breakpoints**

Breakpoints halt instruction execution at particular places in a program. To assure transparent operation, breakpoint control logic is implemented outside the CPU, and particular models of MCU can have different breakpoint capabilities. Refer to the appropriate device manual for detailed information. Generally, breakpoint logic can be configured to halt execution before an instruction executes, or to halt execution on the next instruction boundary following the breakpoint.

### **8.6.1 Breakpoint Type**

There are three basic types of breakpoints:

1. Address-only breakpoints that cause the CPU to execute an SWI. These breakpoints can be set only on addresses. When the breakpoint logic encounters the breakpoint tag, the CPU12 executes an SWI instruction.
2. Address-only breakpoints that cause the MCU to enter BDM. These breakpoints can be set only on addresses. When the breakpoint logic encounters the breakpoint tag, BDM is activated.
3. Address/data breakpoints that cause the MCU to enter BDM. These breakpoints can be set on an address, or on an address and data. When the breakpoint logic encounters the breakpoint tag, BDM is activated.

### **8.6.2 Breakpoint Operation**

Breakpoints use two mechanisms to halt execution:

1. The tag mechanism marks a particular program fetch with a high (even) or low (odd) byte indicator. The tagged byte moves through the instruction queue until a start cycle occurs, then the breakpoint is taken. Breakpoint logic can be configured to force BDM, or to initiate an SWI when the tag is encountered.
2. The force BDM mechanism causes the MCU to enter active BDM at the next instruction start cycle.

CPU12 instructions are used to implement both breakpoint mechanisms. When an SWI tag is encountered, the CPU performs the same sequence of operations as for an SWI. When BDM is forced, the CPU executes a BGND instruction. However, because these operations are not part of the normal flow of instruction execution, the control program must keep track of the actual breakpoint address.

Both SWI and BGND store a return PC value (SWI on the stack and BGND in the CPU12 TMP2 register), but this value is automatically incremented to point to the next instruction after SWI or BGND. In order to resume execution where a breakpoint occurred, the control program must preserve the breakpoint address rather than use the incremented PC value.

The breakpoint logic generally uses match registers to determine when a break is taken. Registers can be used to match the high and low bytes of addresses for single and dual breakpoints, to match data for single breakpoints, or to do both functions. Use of the registers is generally determined by control bit settings.

Both BWP and BOND store a return PC value (BWP on the stack and BOND in the CPU12 TMP2 register), but this value is automatically incremented to point to the next instruction after BWP or BOND, in order to resume execution where a breakpoint occurred; the control program must preserve the breakpoint address rather than use the incremented PC value.

The breakpoint logic generally uses match registers to determine when a break is false. Registers can be used to match the high and low bytes of addresses for single and dual breakpoints, to match data for single breakpoints, or to do both functions. Use of the register is generally determined by control bit settings.



## **SECTION 9**

### **FUZZY LOGIC SUPPORT**

The CPU12 has the first microcontroller instruction set to specifically address the needs of fuzzy logic. This section describes the use of fuzzy logic in control systems, discusses the CPU12 fuzzy logic instructions, and provides examples of fuzzy logic programs.

#### **9.1 Introduction**

The CPU12 includes four instructions that perform specific fuzzy logic tasks. In addition, several other instructions are especially useful in fuzzy logic programs. The overall C-friendliness of the instruction set also aids development of efficient fuzzy logic programs.

This section explains the basic fuzzy logic algorithm for which the four fuzzy logic instructions are intended. Each of the fuzzy logic instructions are then explained in detail. Finally, other custom fuzzy logic algorithms are discussed, with emphasis on use of other CPU12 instructions.

The four fuzzy logic instructions are MEM, which evaluates trapezoidal membership functions; REV and REVW, which perform unweighted or weighted MIN-MAX rule evaluation; and WAV, which performs weighted average defuzzification on singleton output membership functions.

Other instructions that are useful for custom fuzzy logic programs include MINA, EMIND, MAXM, EMAXM, TBL, ETBL, and EMACS. For higher resolution fuzzy programs, the fast extended precision math instructions in the CPU12 are also beneficial. Flexible indexed addressing modes help simplify access to fuzzy logic data structures stored as lists or tabular data structures in memory.

The actual logic additions required to implement fuzzy logic support in the CPU12 are quite small, so there is no appreciable increase in cost for the typical user. A fuzzy inference kernel for the CPU12 requires one-fifth as much code space, and executes fifteen times faster than a comparable kernel implemented on a typical midrange microcontroller. By incorporating fuzzy logic support into a high-volume, general-purpose microcontroller product family, Motorola has made fuzzy logic available for a huge base of applications.

#### **9.2 Fuzzy Logic Basics**

This is an overview of basic fuzzy logic concepts. It can serve as a general introduction to the subject, but that is not the main purpose. There are a number of fuzzy logic programming strategies. This discussion concentrates on the methods implemented in the CPU12 fuzzy logic instructions. The primary goal is to provide a background for a detailed explanation of the CPU12 fuzzy logic instructions.



In general, fuzzy logic provides for set definitions that have fuzzy boundaries rather than the crisp boundaries of Aristotelian logic. These sets can overlap so that, for a specific input value, one or more sets associated with linguistic labels may be true to a degree at the same time. As the input varies from the range of one set into the range of an adjacent set, the first set becomes progressively less true while the second set becomes progressively more true.

Fuzzy logic has membership functions which emulate human concepts like "temperature is warm"; that is, conditions are perceived to have gradual boundaries. This concept seems to be a key element of the human ability to solve certain types of complex problems that have eluded traditional control methods.

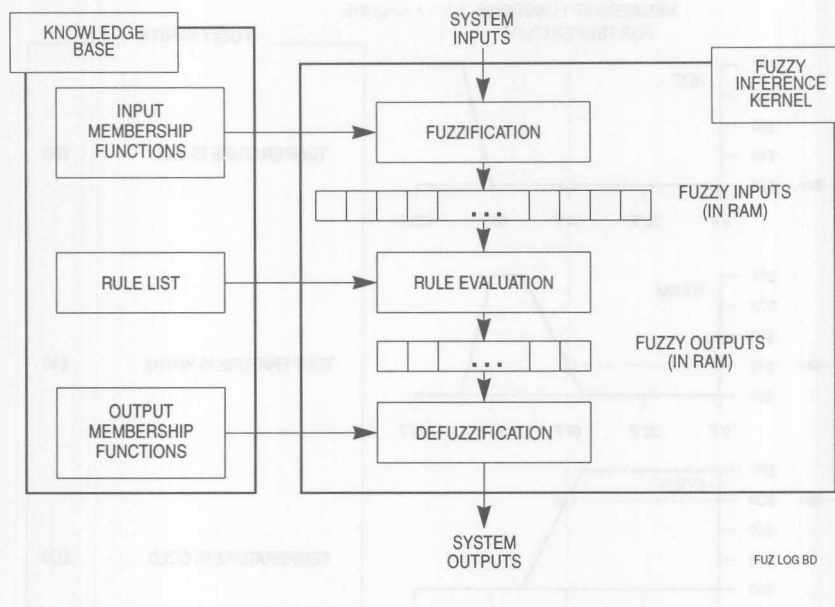
Fuzzy sets provide a means of using linguistic expressions like "temperature is warm" in rules which can then be evaluated with a high degree of numerical precision and repeatability. This directly contradicts the common misperception that fuzzy logic produces approximate results — a specific set of input conditions always produces the same result, just as a conventional control system does.

A microcontroller-based fuzzy logic control system has two parts. The first part is a fuzzy inference kernel which is executed periodically to determine system outputs based on current system inputs. The second part of the system is a knowledge base which contains membership functions and rules. **Figure 9-1** is a block diagram of this kind of fuzzy logic system.

The knowledge base can be developed by an application expert without any microcontroller programming experience. Membership functions are simply expressions of the expert's understanding of the linguistic terms that describe the system to be controlled. Rules are ordinary language statements that describe the actions a human expert would take to solve the application problem.

Rules and membership functions can be reduced to relatively simple data structures (the knowledge base) stored in nonvolatile memory. A fuzzy inference kernel can be written by a programmer who does not know how the application system works. The only thing the programmer needs to do with knowledge base information is store it in the memory locations used by the kernel.

One execution pass through the fuzzy inference kernel generates system output signals in response to current input conditions. The kernel is executed as often as needed to maintain control. If the kernel is executed more often than needed, processor bandwidth and power are wasted; delaying too long between passes can cause the system to get too far out of control. Choosing a periodic rate for a fuzzy control system is the same as it would be for a conventional control system.

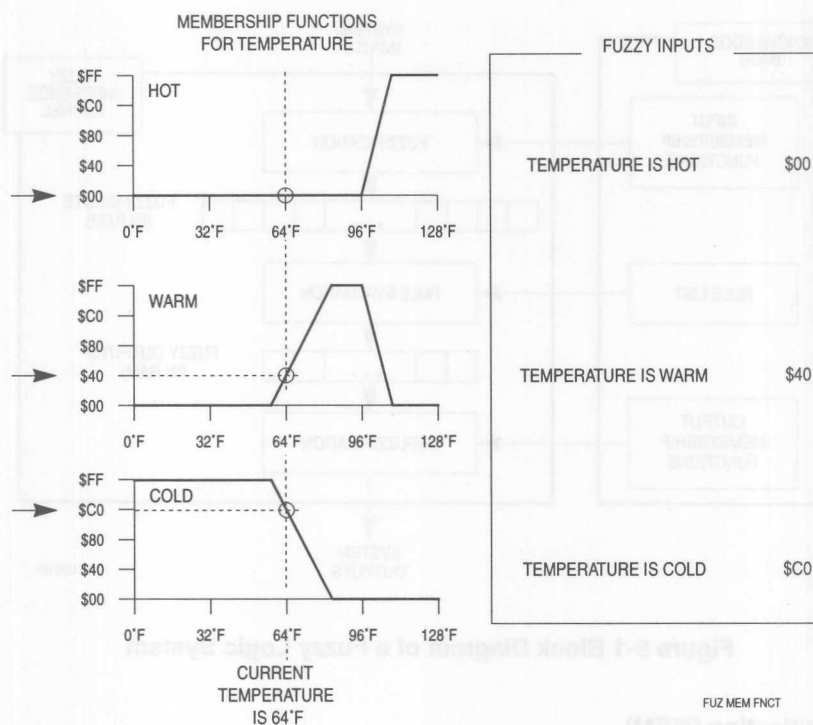


**Figure 9-1 Block Diagram of a Fuzzy Logic System**

### 9.2.1 Fuzzification (MEM)

During the fuzzification step, the current system input values are compared against stored input membership functions to determine the degree to which each label of each system input is true. This is accomplished by finding the y-value for the current input value on a trapezoidal membership function for each label of each system input. The MEM instruction in the CPU12 performs this calculation for one label of one system input. To perform the complete fuzzification task for a system, several MEM instructions must be executed, usually in a program loop structure.

**Figure 9-2** shows a system of three input membership functions, one for each label of the system input. The x-axis of all three membership functions represents the range of possible values of the system input. The vertical line through all three membership functions represents a specific system input value. The y-axis represents degree of truth and varies from completely false (\$00 or 0%) to completely true (\$FF or 100%). The y-value where the vertical line intersects each of the membership functions, is the degree to which the current input value matches the associated label for this system input. For example, the expression "temperature is warm" is 25% true (\$40). The value \$40 is stored to a RAM location, and is called a fuzzy input (in this case, the fuzzy input for "temperature is warm"). There is a RAM location for each fuzzy input (for each label of each system input).



**Figure 9-2 Fuzzification Using Membership Functions**

When the fuzzification step begins, the current value of the system input is in an accumulator of the CPU12, one index register points to the first membership function definition in the knowledge base, and a second index register points to the first fuzzy input in RAM. As each fuzzy input is calculated by executing a MEM instruction, the result is stored to the fuzzy input and both pointers are updated automatically to point to the locations associated with the next fuzzy input. The MEM instruction takes care of everything except counting the number of labels per system input and loading the current value of any subsequent system inputs.

The end result of the fuzzification step is a table of fuzzy inputs representing current system conditions.

### 9.2.2 Rule Evaluation (REV and REVW)

Rule evaluation is the central element of a fuzzy logic inference program. This step processes a list of rules from the knowledge base using current fuzzy input values from RAM to produce a list of fuzzy outputs in RAM. These fuzzy outputs can be thought of as raw suggestions for what the system output should be in response to the current input conditions. Before the results can be applied, the fuzzy outputs must be further processed, or defuzzified, to produce a single output value that represents the combined effect of all of the fuzzy outputs.

The CPU12 offers two variations of rule evaluation instructions. The REV instruction provides for unweighted rules (all rules are considered to be equally important). The REVW instruction is similar but allows each rule to have a separate weighting factor which is stored in a separate parallel data structure in the knowledge base. In addition to the weights, the two rule evaluation instructions also differ in the way rules are encoded into the knowledge base.

An understanding of the structure and syntax of rules is needed to understand how a microcontroller performs the rule evaluation task. The following is an example of a typical rule.

If temperature is warm and pressure is high then heat is (should be) off.

At first glance, it seems that encoding this rule in a compact form understandable to the microcontroller would be difficult, but it is actually simple to reduce the rule to a small list of memory pointers. The left portion of the rule is a statement of input conditions and the right portion of the rule is a statement of output actions.

The left portion of a rule is made up of one or more (in this case two) antecedents connected by a fuzzy *and* operator. Each antecedent expression consists of the name of a system input, followed by *is*, followed by a label name. The label must be defined by a membership function in the knowledge base. Each antecedent expression corresponds to one of the fuzzy inputs in RAM. Since *and* is the only operator allowed to connect antecedent expressions, there is no need to include these in the encoded rule. The antecedents can be encoded as a simple list of pointers to (or addresses of) the fuzzy inputs to which they refer.

The right portion of a rule is made up of one or more (in this case one) consequents. Each consequent expression consists of the name of a system output, followed by *is*, followed by a label name. Each consequent expression corresponds to a specific fuzzy output in RAM. Consequents for a rule can be encoded as a simple list of pointers to (or addresses of) the fuzzy outputs to which they refer.

The complete rules are stored in the knowledge base as a list of pointers or addresses of fuzzy inputs and fuzzy outputs. In order for the rule evaluation logic to work, there must be some means of knowing which pointers refer to fuzzy inputs, and which refer to fuzzy outputs. There also must be a way to know when the last rule in the system has been reached.

One method of organization is to have a fixed number of rules with a specific number of antecedents and consequents. A second method, employed in Motorola Freeware M68HC11 kernels, is to mark the end of the rule list with a reserved value, and use a bit in the pointers to distinguish antecedents from consequents. A third method of organization, used in the CPU12, is to mark the end of the rule list with a reserved value, and separate antecedents and consequents with another reserved value. This permits any number of rules, and allows each rule to have any number of antecedents and consequents, subject to the limits imposed by availability of system memory.

Each rule is evaluated sequentially, but the rules as a group are treated as if they were all evaluated simultaneously. Two mathematical operations take place during rule evaluation. The fuzzy *and* operator corresponds to the mathematical minimum operation and the fuzzy *or* operation corresponds to the mathematical maximum operation. The fuzzy *and* is used to connect antecedents within a rule. The fuzzy *or* is implied between successive rules. Before evaluating any rules, all fuzzy outputs are set to zero (meaning not true at all). As each rule is evaluated, the smallest (minimum) antecedent is taken to be the overall truth of the rule. This rule truth value is applied to each consequent of the rule (by storing this value to the corresponding fuzzy output) unless the fuzzy output is already larger (maximum). If two rules affect the same fuzzy output, the rule that is most true governs the value in the fuzzy output because the rules are connected by an implied fuzzy *or*.

In the case of rule weighting, the truth value for a rule is determined as usual by finding the smallest rule antecedent. Before applying this truth value to the consequents for the rule, the value is multiplied by a fraction from zero (rule disabled) to one (rule fully enabled). The resulting modified truth value is then applied to the fuzzy outputs.

The end result of the rule evaluation step is a table of suggested or "raw" fuzzy outputs in RAM. These values were obtained by plugging current conditions (fuzzy input values) into the system rules in the knowledge base. The raw results cannot be supplied directly to the system outputs because they may be ambiguous. For instance, one raw output can indicate that the system output should be medium with a degree of truth of 50% while, at the same time, another indicates that the system output should be low with a degree of truth of 25%. The defuzzification step resolves these ambiguities.

### 9.2.3 Defuzzification (WAV)

The final step in the fuzzy logic program combines the raw fuzzy outputs into a composite system output. Unlike the trapezoidal shapes used for inputs, the CPU12 typically uses singletons for output membership functions. As with the inputs, the x-axis represents the range of possible values for a system output. Singleton membership functions consist of the x-axis position for a label of the system output. Fuzzy outputs correspond to the y-axis height of the corresponding output membership function.

The WAV instruction calculates the numerator and denominator sums for weighted average of the fuzzy outputs according to the formula:

$$\text{System Output} = \frac{\sum_{i=1}^n S_i F_i}{\sum_{i=1}^n F_i}$$

Where  $n$  is the number of labels of a system output,  $S_i$  are the singleton positions from the knowledge base, and  $F_i$  are fuzzy outputs from RAM. For a common fuzzy logic program on the CPU12,  $n$  is eight or less (though this instruction can handle any value to 255) and  $S_i$  and  $F_i$  are 8-bit values. The final divide is performed with a separate EDIV instruction placed immediately after the WAV instruction.

Before executing WAV, an accumulator must be loaded with the number of iterations ( $n$ ), one index register must be pointed at the list of singleton positions in the knowledge base, and a second index register must be pointed at the list of fuzzy outputs in RAM. If the system has more than one system output, the WAV instruction is executed once for each system output.

### 9.3 Example Inference Kernel

**Figure 9-3** is a complete fuzzy inference kernel written in CPU12 assembly language. Numbers in square brackets are cycle counts. The kernel uses two system inputs with seven labels each and one system output with seven labels. The program assembles to 57 bytes. It executes in about 54  $\mu\text{s}$  at an 8 MHz bus rate. The basic structure can easily be extended to a general-purpose system with a larger number of inputs and outputs.

Lines 1 to 3 set up pointers and load the system input value into the A accumulator.

Line 4 sets the loop count for the loop in lines 5 and 6.

Lines 5 and 6 make up the fuzzification loop for seven labels of one system input. The MEM instruction finds the y-value on a trapezoidal membership function for the current input value, for one label of the current input, and then stores the result to the corresponding fuzzy input. Pointers in X and Y are automatically updated by four and one so they point at the next membership function and fuzzy input respectively.

Line 7 loads the current value of the next system input. Pointers in X and Y already point to the right places as a result of the automatic update function of the MEM instruction in line 5.

Line 8 reloads a loop count.

Lines 9 and 10 form a loop to fuzzify the seven labels of the second system input. When the program drops to line 11, the Y index register is pointing at the next location after the last fuzzy input, which is the first fuzzy output in this system.



```

*
01 [2] FUZZIFY LDX #INPUT_MFS ;Point at MF definitions
02 [2] LDY #FUZ_INS ;Point at fuzzy input table
03 [3] LDAA CURRENT_INS ;Get first input value
04 [1] LDAB #7 ;7 labels per input
05 [5] GRAD_LOOP MEM ;Evaluate one MF
06 [3] DBNE B,GRAD_LOOP ;For 7 labels of 1 input
07 [3] LDAA CURRENT_INS+1 ;Get second input value
08 [1] LDAB #7 ;7 labels per input
09 [5] GRAD_LOOP1 MEM ;Evaluate one MF
10 [3] DBNE B,GRAD_LOOP1 ;For 7 labels of 1 input

11 [1] LDAB #7 ;Loop count
12 [2] RULE_EVAL CLR 1,Y+ ;Clr a fuzzy out & inc ptr
13 [3] DBNE b,RULE_EVAL ;Loop to clr all fuzzy outs
14 [2] LDX #RULE_START ;Point at first rule element
15 [2] LDY #FUZ_INS ;Point at fuzzy ins and outs
16 [1] LDAA #$FF ;Init A (and clears V-bit)
17 [3n+4] REV ;Process rule list

18 [2] DEFUZ LDY #FUZ_OUT ;Point at fuzzy outputs
19 [1] LDX #SGLTN_POS ;Point at singleton positions
20 [1] LDAB #7 ;7 fuzzy outs per COG output
21 [8b+9] WAV ;Calculate sums for wtd av
22 [11] EDIV ;Final divide for wtd av
23 [1] TFR Y D ;Move result to A:B
24 [3] STAB COG_OUT ;Store system output
*
***** End

```

**Figure 9-3 Fuzzy Inference Engine**

Line 11 sets the loop count to clear seven fuzzy outputs.

Lines 12 and 13 form a loop to clear all fuzzy outputs before rule evaluation starts.

Line 14 initializes the X index register to point at the first element in the rule list for the REV instruction.

Line 15 initializes the Y index register to point at the fuzzy inputs and outputs in the system. The rule list (for REV) consists of 8-bit offsets from this base address to particular fuzzy inputs or fuzzy outputs. The special value \$FE is interpreted by REV as a marker between rule antecedents and consequents.

Line 16 initializes the A accumulator to the highest 8-bit value in preparation for finding the smallest fuzzy input referenced by a rule antecedent. The LDAA #\$FF instruction also clears the V-bit in the CPU12's condition code register so the REV instruction knows it is processing antecedents. During rule list processing, the V bit is toggled each time an \$FE is detected in the list. The V bit indicates whether REV is processing antecedents or consequents.

Line 17 is the REV instruction, a self-contained loop to process successive elements in the rule list until an \$FF character is found. For a system of 17 rules with two antecedents and one consequent each, the REV instruction takes 259 cycles, but it is interruptible so it does not cause a long interrupt latency.

Lines 18 through 20 set up pointers and an iteration count for the WAV instruction.

Line 21 is the beginning of defuzzification. The WAV instruction calculates a sum-of-products and a sum-of-weights.

Line 22 completes defuzzification. The EDIV instruction performs a 32-bit by 16-bit divide on the intermediate results from WAV to get the weighted average.

Line 23 moves the EDIV result into the double accumulator.

Line 24 stores the low 8-bits of the defuzzification result.

This example inference program shows how easy it is to incorporate fuzzy logic into general applications using the CPU12. Code space and execution time are no longer serious factors in the decision to use fuzzy logic. The next section begins a much more detailed look at the fuzzy logic instructions of the CPU12.

#### 9.4 MEM Instruction Details

This section provides a more detailed explanation of the membership function evaluation instruction (MEM), including details about abnormal special cases for improperly defined membership functions.

##### 9.4.1 Membership Function Definitions

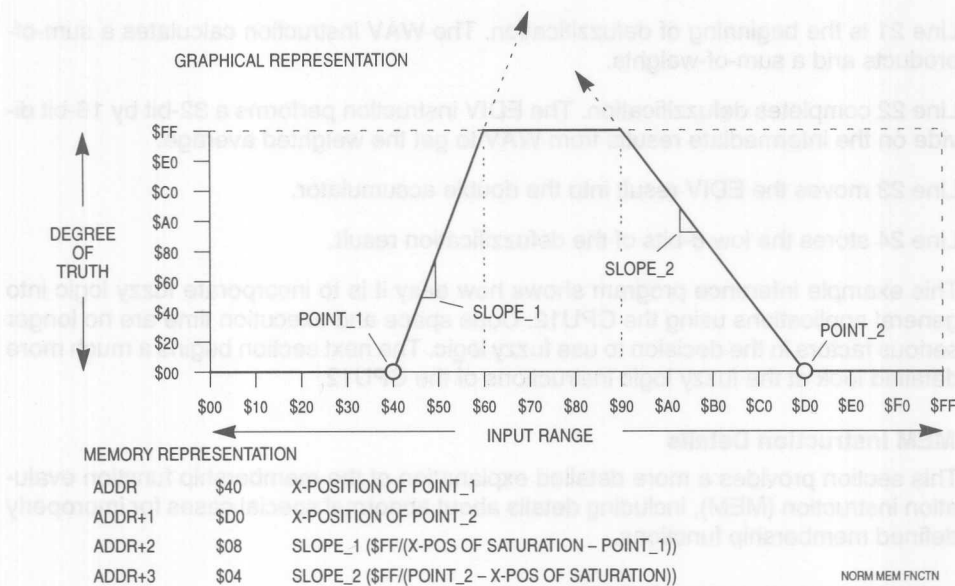
**Figure 9-4** shows how a normal membership function is specified in the CPU12. Typically a software tool is used to input membership functions graphically, and the tool generates data structures for the target processor and software kernel. Alternatively, points and slopes for the membership functions can be determined and stored in memory with define-constant assembler directives.

An internal CPU algorithm calculates the y-value where the current input intersects a membership function. This algorithm assumes the membership function obeys some common-sense rules. If the membership function definition is improper, the results may be unusual. **9.4.2 Abnormal Membership Function Definitions** discusses these cases. The following rules apply to normal membership functions.

- $\$00 \leq \text{point1} < \$FF$
- $\$00 < \text{point2} \leq \$FF$
- $\text{point1} < \text{point2}$
- The sloping sides of the trapezoid meet at or above  $\$FF$

Each system input such as temperature has several labels such as cold, cool, normal, warm, and hot. Each label of each system input must have a membership function to describe its meaning in an unambiguous numerical way. Typically, there are three to seven labels per system input, but there is no practical restriction on this number as far as the fuzzification step is concerned.





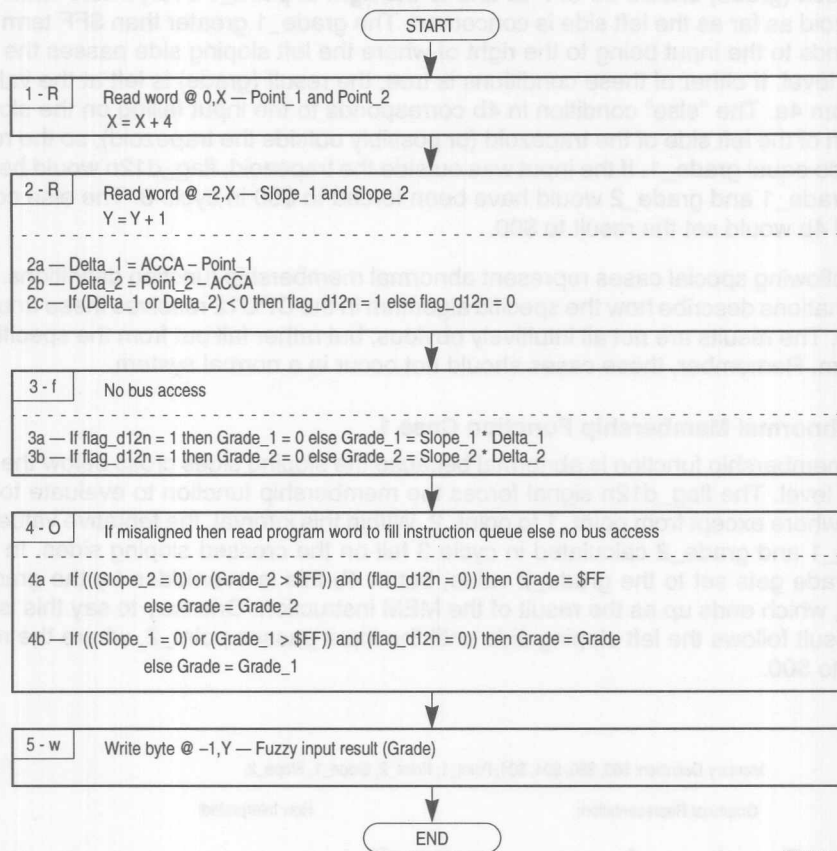
**Figure 9-4 Defining a Normal Membership Function**

#### 9.4.2 Abnormal Membership Function Definitions

In the CPU12, it is possible (and proper) to define “crisp” membership functions. A crisp membership function has one or both sides vertical (infinite slope). Since the slope value \$00 is not used otherwise, it is assigned to mean infinite slope to the MEM instruction in the CPU12.

Although a good fuzzy development tool will not allow the user to specify an improper membership function, it is possible to have program errors or memory errors which result in erroneous abnormal membership functions. Although these abnormal shapes do not correspond to any working systems, understanding how the CPU12 treats these cases can be helpful for debugging.

A close examination of the MEM instruction algorithm will show how such membership functions are evaluated. **Figure 9-5** is a complete flow diagram for the execution of a MEM instruction. Each rectangular box represents one CPU bus cycle. The number in the upper left corner corresponds to the cycle number and the letter corresponds to the cycle type (refer to **SECTION 6 INSTRUCTION GLOSSARY** for details). The upper portion of the box includes information about bus activity during this cycle (if any). The lower portion of the box, which is separated by a dashed line, includes information about internal CPU processes. It is common for several internal functions to take place during a single CPU cycle (for example, in cycle 2, two 8-bit subtractions take place and a flag is set based on the results).



**Figure 9-5 MEM Instruction Flow Diagram**

Consider 4a: If (((Slope\_2 = 0) or (Grade\_2 > \$FF)) and (flag\_d12n = 0)).

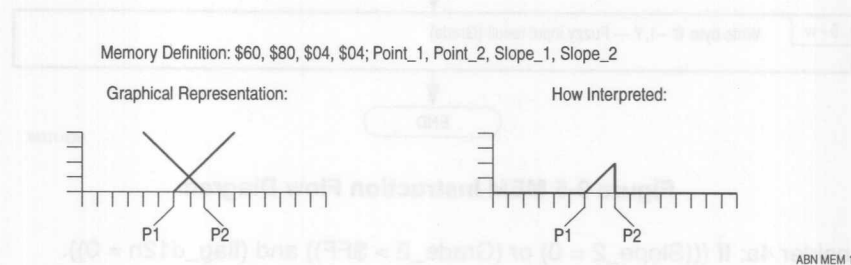
The flag\_d12n is zero as long as the input value (in accumulator A) is within the trapezoid. Everywhere outside the trapezoid, one or the other delta term will be negative, and the flag will equal one. Slope\_2 equals zero indicates the right side of the trapezoid has infinite slope, so the resulting grade should be \$FF everywhere in the trapezoid, including at point\_2, as far as this side is concerned. The term grade\_2 greater than \$FF means the value is far enough into the trapezoid that the right sloping side of the trapezoid has crossed above the \$FF cutoff level and the resulting grade should be \$FF as far as the right sloping side is concerned. 4a decides if the value is left of the right sloping side (Grade = \$FF), or on the sloping portion of the right side of the trapezoid (Grade = Grade\_2). 4b could still override this tentative value in grade.

In 4b, slope\_1 is zero if the left side of the trapezoid has infinite slope (vertical). If so, the result (grade) should be \$FF at and to the right of point\_1 everywhere within the trapezoid as far as the left side is concerned. The grade\_1 greater than \$FF term corresponds to the input being to the right of where the left sloping side passes the \$FF cutoff level. If either of these conditions is true, the result (grade) is left at the value it got from 4a. The "else" condition in 4b corresponds to the input falling on the sloping portion of the left side of the trapezoid (or possibly outside the trapezoid), so the result is grade equal grade\_1. If the input was outside the trapezoid, flag\_d12n would be one and grade\_1 and grade\_2 would have been forced to \$00 in cycle 3. The else condition of 4b would set the result to \$00.

The following special cases represent abnormal membership function definitions. The explanations describe how the specific algorithm in the CPU12 resolves these unusual cases. The results are not all intuitively obvious, but rather fall out from the specific algorithm. Remember, these cases should not occur in a normal system.

#### 9.4.2.1 Abnormal Membership Function Case 1

This membership function is abnormal because the sloping sides cross below the \$FF cutoff level. The flag\_d12n signal forces the membership function to evaluate to \$00 everywhere except from point\_1 to point\_2. Within this interval, the tentative values for grade\_1 and grade\_2 calculated in cycle 3 fall on the crossed sloping sides. In step 4a, grade gets set to the grade\_2 value, but in 4b this is overridden by the grade\_1 value, which ends up as the result of the MEM instruction. One way to say this is that the result follows the left sloping side until the input passes point\_2, where the result goes to \$00.



**Figure 9-6 Abnormal Membership Function Case 1**

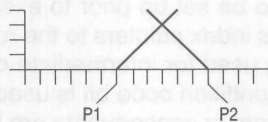
If point\_1 was to the right of point\_2, flag\_d12n would force the result to be \$00 for all input values. In fact, flag\_d12n always limits the region of interest to the space greater than or equal to point\_1 and less than or equal to point\_2.

### 9.4.2.2 Abnormal Membership Function Case 2

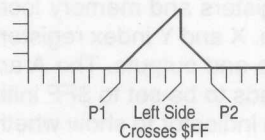
Like the previous example, the membership function in case 2 is abnormal because the sloping sides cross below the \$FF cutoff level, but the left sloping side reaches the \$FF cutoff level before the input gets to point\_2. In this case, the result follows the left sloping side until it reaches the \$FF cutoff level. At this point, the (grade\_1 > \$FF) term of 4b kicks in, making the expression true so grade equals grade (no overwrite). The result from here to point\_2 becomes controlled by the "else" part of 4a (grade = grade\_2), and the result follows the right sloping side.

Memory Definition: \$60, \$C0, \$04, \$04; Point\_1, Point\_2, Slope\_1, Slope\_2

Graphical Representation



How Interpreted



ABN MEM 2

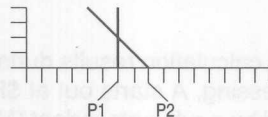
Figure 9-7 Abnormal Membership Function Case 2

### 9.4.2.3 Abnormal Membership Function Case 3

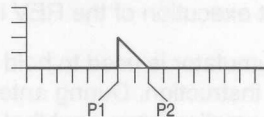
The membership function in case 3 is abnormal because the sloping sides cross below the \$FF cutoff level, and the left sloping side has infinite slope. In this case, 4a is not true, so grade equals grade\_2. 4b is true because slope\_1 is zero, so 4b does not overwrite grade.

Memory Definition: \$60, \$80, \$00, \$04; Point\_1, Point\_2, Slope\_1, Slope\_2

Graphical Representation



How Interpreted



ABN MEM 3

Figure 9-8 Abnormal Membership Function Case 3

## 9.5 REV, REVW Instruction Details

This section provides a more detailed explanation of the rule evaluation instructions (REV and REVW). The data structures used to specify rules are somewhat different for the weighted versus unweighted versions of the instruction. One uses 8-bit offsets in the encoded rules, while the other uses full 16-bit addresses. This affects the size of the rule data structure and execution time.

### 9.5.1 Unweighted Rule Evaluation (REV)

This instruction implements basic min-max rule evaluation. CPU registers are used for pointers and intermediate calculation results.

Since the REV instruction is essentially a list-processing instruction, execution time is dependent on the number of elements in the rule list. The REV instruction is interruptible (typically within three bus cycles), so it does not adversely affect worst case interrupt latency. Since all intermediate results and instruction status are held in stacked CPU registers, the interrupt service code can even include independent REV and REVW instructions.

#### 9.5.1.1 Set Up Prior to Executing REV

Some CPU registers and memory locations need to be set up prior to executing the REV instruction. X and Y index registers are used as index pointers to the rule list and the fuzzy inputs and outputs. The A accumulator is used for intermediate calculation results and needs to be set to \$FF initially. The V condition code bit is used as an instruction status indicator to show whether antecedents or consequents are being processed. Initially, the V bit is cleared to zero to indicate antecedents are being processed. The fuzzy outputs (working RAM locations) need to be cleared to \$00. If these values are not initialized before executing the REV instruction, results will be erroneous.

The X index register is set to the address of the first element in the rule list (in the knowledge base). The REV instruction automatically updates this pointer so that the instruction can resume correctly if it is interrupted. After the REV instruction finishes, X will point at the next address past the \$FF separator character that marks the end of the rule list.

The Y index register is set to the base address for the fuzzy inputs and outputs (in working RAM). Each rule antecedent is an unsigned 8-bit offset from this base address to the referenced fuzzy input. Each rule consequent is an unsigned 8-bit offset from this base address to the referenced fuzzy output. The Y index register remains constant throughout execution of the REV instruction.

The 8-bit A accumulator is used to hold intermediate calculation results during execution of the REV instruction. During antecedent processing, A starts out at \$FF and is replaced by any smaller fuzzy input that is referenced by a rule antecedent (MIN). During consequent processing, A holds the truth value for the rule. This truth value is stored to any fuzzy output that is referenced by a rule consequent, unless that fuzzy output is already larger (MAX).

Before starting to execute REV, A must be set to \$FF (the largest 8-bit value) because rule evaluation always starts with processing of the antecedents of the first rule. For subsequent rules in the list, A is automatically set to \$FF when the instruction detects the \$FE marker character between the last consequent of the previous rule, and the first antecedent of a new rule.

The instruction LDAA #\$FF clears the V bit at the same time it initializes A to \$FF. This satisfies the REV setup requirement to clear the V bit as well as the requirement to initialize A to \$FF. Once the REV instruction starts, the value in the V bit is automatically maintained as \$FE separator characters are detected.

The final requirement to clear all fuzzy outputs to \$00 is part of the MAX algorithm. Each time a rule consequent references a fuzzy output, that fuzzy output is compared to the truth value for the current rule. If the current truth value is larger, it is written over the previous value in the fuzzy output. After all rules have been evaluated, the fuzzy output contains the truth value for the most-true rule that referenced that fuzzy output.

After REV finishes, A will hold the truth value for the last rule in the rule list. The V condition code bit should be one because the last element before the \$FF end marker should have been a rule consequent. If V is zero after executing REV, it indicates the rule list was structured incorrectly.

#### 9.5.1.2 Interrupt Details

The REV instruction includes a three-cycle processing loop for each byte in the rule list (including antecedents, consequents, and special separator characters). Within this loop, a check is performed to see if any qualified interrupt request is pending. If an interrupt is detected, the current CPU registers are stacked and the interrupt is honored. When the interrupt service routine finishes, an RTI instruction causes the CPU to recover its previous context from the stack, and the REV instruction is resumed as if it had not been interrupted.

The stacked value of the program counter (PC), in case of an interrupted REV instruction, points to the REV instruction rather than the instruction that follows. This causes the CPU to try to execute a new REV instruction upon return from the interrupt. Since the CPU registers (including the V bit in the condition codes register) indicate the current status of the interrupted REV instruction, this effectively causes the rule evaluation operation to resume from where it left off.

#### 9.5.1.3 Cycle-by-Cycle Details for REV

The central element of the REV instruction is a three-cycle loop that is executed once for each byte in the rule list. There is a small amount of housekeeping activity to get this loop started as REV begins, and a small sequence to end the instruction. If an interrupt comes, there is a special small sequence to save CPU status on the stack before honoring the requested interrupt.

**Figure 9-9** is a REV instruction flow diagram. Each rectangular box represents one CPU clock cycle. Decision blocks and connecting arrows are considered to take no time at all. The letters in the small rectangles in the upper left corner of each bold box correspond to execution cycle codes (refer to **SECTION 6 INSTRUCTION GLOSSARY** for details). Lower case letters indicate a cycle where 8-bit or no data is transferred. Upper case letters indicate cycles where 16-bit or no data is transferred.



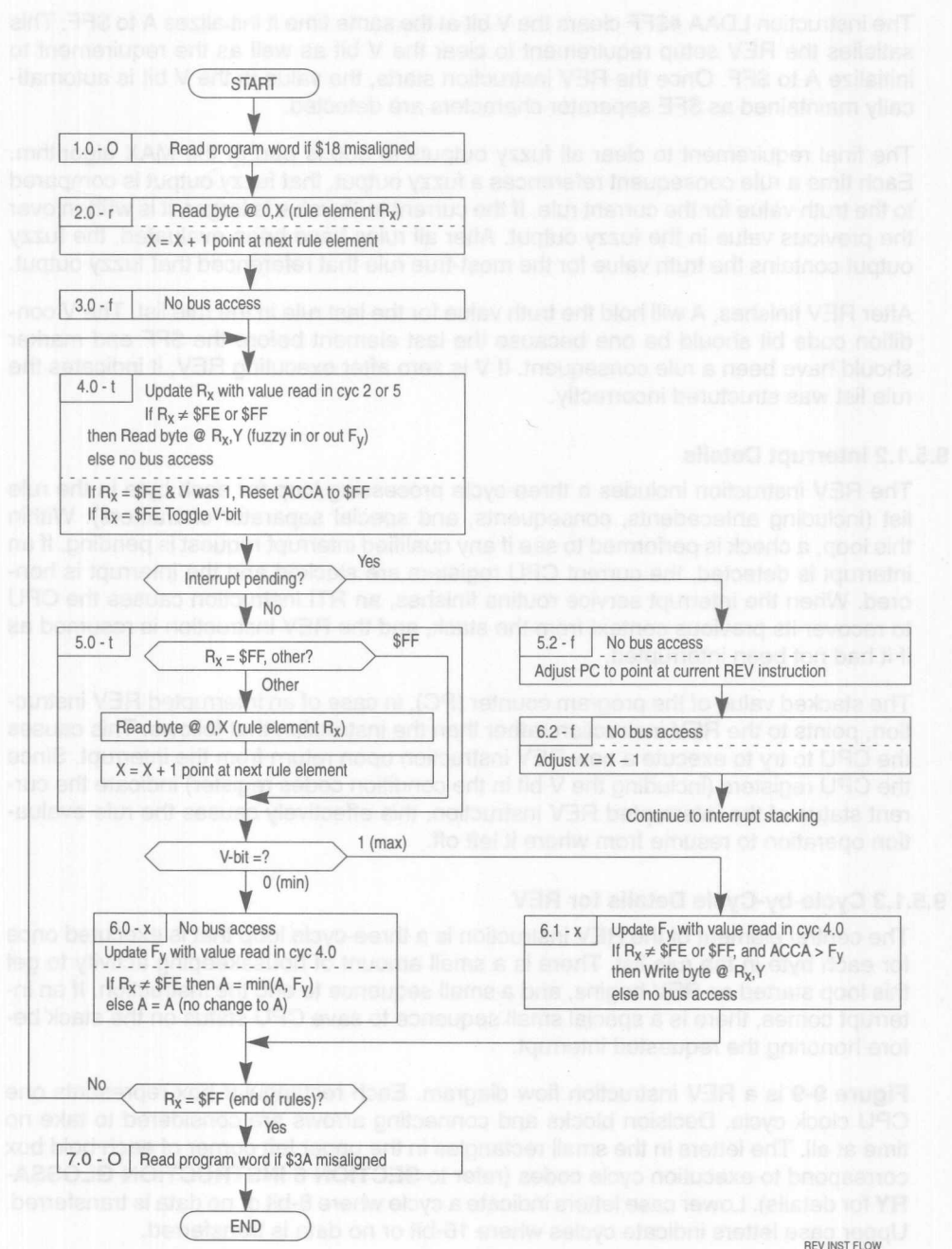


Figure 9-9 REV Instruction Flow Diagram

When a value is read from memory, it cannot be used by the CPU until the second cycle after the read takes place. This is due to access and propagation delays.

Since there is more than one flow path through the REV instruction, cycle numbers have a decimal place. This decimal place indicates which of several possible paths is being used. The CPU normally moves forward by one digit at a time within the same flow (flow number is indicated after the decimal point in the cycle number). There are two exceptions possible to this orderly sequence through an instruction. The first is a branch back to an earlier cycle number to form a loop as in 6.0 to 4.0. The second type of sequence change is from one flow to a parallel flow within the same instruction such as 4.0 to 5.2, which occurs if the REV instruction senses an interrupt. In this second type of sequence branch, the whole number advances by one and the flow number changes to a new value (the digit after the decimal point).

In cycle 1.0, the CPU12 does an optional program word access to replace the \$18 prebyte of the REV instruction. Notice that cycle 7.0 is also an O type cycle. One or the other of these will be a program word fetch, while the other will be a free cycle where the CPU does not access the bus. Although the \$18 page prebyte is a required part of the REV instruction, it is treated by the CPU12 as a somewhat separate single cycle instruction.

Rule evaluation begins at cycle 2.0 with a byte read of the first element in the rule list. Usually this would be the first antecedent of the first rule, but the REV instruction can be interrupted, so this could be a read of any byte in the rule list. The X index register is incremented so it points to the next element in the rule list. Cycle 3.0 is needed to satisfy the required delay between a read and when data is valid to the CPU. Some internal CPU housekeeping activity takes place during this cycle, but there is no bus activity. By cycle 4.0, the rule element that was read in cycle 2.0 is available to the CPU.

Cycle 4.0 is the first cycle of the main three cycle rule evaluation loop. Depending upon whether rule antecedents or consequents are being processed, the loop will consist of cycles 4.0, 5.0, 6.0, or the sequence 4.0, 5.0, 6.1. This loop is executed once for every byte in the rule list, including the \$FE separators and the \$FF end-of-rules marker.

At each cycle 4.0, a fuzzy input or fuzzy output is read, except during the loop passes associated with the \$FE and \$FF marker bytes, where no bus access takes place during cycle 4.0. The read access uses the Y index register as the base address and the previously read rule byte ( $R_x$ ) as an unsigned offset from Y. The fuzzy input or output value read here will be used during the next cycle 6.0 or 6.1. Besides being used as the offset from Y for this read, the previously read  $R_x$  is checked to see if it is a separator character (\$FE). If  $R_x$  was \$FE and the V-bit was one, this indicates a switch from processing consequents of one rule to starting to process antecedents of the next rule. At this transition, the A accumulator is initialized to \$FF to prepare for the min operation to find the smallest fuzzy input. Also, if  $R_x$  is \$FE, the V-bit is toggled to indicate the change from antecedents to consequents, or consequents to antecedents.

During cycle 5.0, a new rule byte is read unless this is the last loop pass, and  $R_x$  is \$FF (marking the end of the rule list). This new rule byte will not be used until cycle 4.0 of the next pass through the loop.



Between cycle 5.0 and 6.x, the V-bit is used to decide which of two paths to take. If V is zero, antecedents are being processed and the CPU progresses to cycle 6.0. If V is one, consequents are being processed and the CPU goes to cycle 6.1.

During cycle 6.0, the current value in the A accumulator is compared to the fuzzy input that was read in the previous cycle 4.0, and the lower value is placed in the A accumulator (min operation). If Rx is \$FE, this is the transition between rule antecedents and rule consequents, and this min operation is skipped (although the cycle is still used). No bus access takes place during cycle 6.0 but cycle 6.x is considered an x type cycle because it could be a byte write (cycle 6.1), or a free cycle (cycle 6.0 or 6.1 with Rx = \$FE or \$FF).

If an interrupt arrives while the REV instruction is executing, REV can break between cycles 4.0 and 5.0 in an orderly fashion so that the rule evaluation operation can resume after the interrupt has been serviced. Cycles 5.2 and 6.2 are needed to adjust the PC and X index register so the REV operation can recover after the interrupt. PC is adjusted backward in cycle 5.2 so it points to the currently running REV instruction. After the interrupt, rule evaluation will resume, but the values that were stored on the stack for index registers, accumulator A, and CCR will cause the operation to pick up where it left off. In cycle 6.2, the X index register is adjusted backward by one because the last rule byte needs to be re-fetched when the REV instruction resumes.

After cycle 6.2, the REV instruction is finished, and execution would continue to the normal interrupt processing flow.

### 9.5.2 Weighted Rule Evaluation (RE VW)

This instruction implements a weighted variation of min-max rule evaluation. The weighting factors are stored in a table with one 8-bit entry per rule. The weight is used to multiply the truth value of the rule (minimum of all antecedents) by a value from zero to one to get the weighted result. This weighted result is then applied to the consequents, just as it would be for unweighted rule evaluation.

Since the RE VW instruction is essentially a list-processing instruction, execution time is dependent on the number of rules and the number of elements in the rule list. The RE VW instruction is interruptible (typically within three to five bus cycles), so it does not adversely affect worst case interrupt latency. Since all intermediate results and instruction status are held in stacked CPU registers, the interrupt service code can even include independent REV and RE VW instructions.

The rule structure is different for RE VW than for REV. For RE VW, the rule list is made up of 16-bit elements rather than 8-bit elements. Each antecedent is represented by the full 16-bit address of the corresponding fuzzy input. Each rule consequent is represented by the full address of the corresponding fuzzy output.

The markers separating antecedents from consequents are the reserved 16-bit value \$FFFE, and the end of the last rule is marked by the reserved 16-bit value \$FFFF. Since \$FFFE and \$FFFF correspond to the addresses of the reset vector, there would never be a fuzzy input or output at either of these locations.

#### 9.5.2.1 Set Up Prior to Executing REVW

Some CPU registers and memory locations need to be set up prior to executing the REVW instruction. X and Y index registers are used as index pointers to the rule list and the list of rule weights. The A accumulator is used for intermediate calculation results and needs to be set to \$FF initially. The V condition code bit is used as an instruction status indicator that shows whether antecedents or consequents are being processed. Initially the V bit is cleared to zero to indicate antecedents are being processed. The C condition code bit is used to indicate whether rule weights are to be used (1) or not (0). The fuzzy outputs (working RAM locations) need to be cleared to \$00. If these values are not initialized before executing the REVW instruction, results will be erroneous.

The X index register is set to the address of the first element in the rule list (in the knowledge base). The REVW instruction automatically updates this pointer so that the instruction can resume correctly if it is interrupted. After the REVW instruction finishes, X will point at the next address past the \$FFFF separator word that marks the end of the rule list.

The Y index register is set to the starting address of the list of rule weights. Each rule weight is an 8-bit value. The weighted result is the truncated upper eight bits of the 16-bit result, which is derived by multiplying the minimum rule antecedent value (\$00–\$FF) by the weight plus one (\$001–\$100). This method of weighting rules allows an 8-bit weighting factor to represent a value between zero and one inclusive.

The 8-bit A accumulator is used to hold intermediate calculation results during execution of the REVW instruction. During antecedent processing, A starts out at \$FF and is replaced by any smaller fuzzy input that is referenced by a rule antecedent. If rule weights are enabled by the C condition code bit equal one, the rule truth value is multiplied by the rule weight just before consequent processing starts. During consequent processing, A holds the truth value (possibly weighted) for the rule. This truth value is stored to any fuzzy output that is referenced by a rule consequent, unless that fuzzy output is already larger (MAX).

Before starting to execute REVW, A must be set to \$FF (the largest 8-bit value) because rule evaluation always starts with processing of the antecedents of the first rule. For subsequent rules in the list, A is automatically set to \$FF when the instruction detects the \$FFFE marker word between the last consequent of the previous rule, and the first antecedent of a new rule.

Both the C and V condition code bits must be set up prior to starting a REVW instruction. Once the REVW instruction starts, the C bit remains constant and the value in the V bit is automatically maintained as \$FFFE separator words are detected.

The final requirement to clear all fuzzy outputs to \$00 is part of the MAX algorithm. Each time a rule consequent references a fuzzy output, that fuzzy output is compared to the truth value (weighted) for the current rule. If the current truth value is larger, it is written over the previous value in the fuzzy output. After all rules have been evaluated, the fuzzy output contains the truth value for the most-true rule that referenced that fuzzy output.

After REVW finishes, A will hold the truth value (weighted) for the last rule in the rule list. The V condition code bit should be one because the last element before the \$FFFF end marker should have been a rule consequent. If V is zero after executing REVW, it indicates the rule list was structured incorrectly.

#### 9.5.2.2 Interrupt Details

The REVW instruction includes a three-cycle processing loop for each word in the rule list (this loop expands to five cycles between antecedents and consequents to allow time for the multiplication with the rule weight). Within this loop, a check is performed to see if any qualified interrupt request is pending. If an interrupt is detected, the current CPU registers are stacked and the interrupt is honored. When the interrupt service routine finishes, an RTI instruction causes the CPU to recover its previous context from the stack, and the REVW instruction is resumed as if it had not been interrupted.

The stacked value of the program counter (PC), in case of an interrupted REVW instruction, points to the REVW instruction rather than the instruction that follows. This causes the CPU to try to execute a new REVW instruction upon return from the interrupt. Since the CPU registers (including the C bit and V bit in the condition codes register) indicate the current status of the interrupted REVW instruction, this effectively causes the rule evaluation operation to resume from where it left off.

#### 9.5.2.3 Cycle-by-Cycle Details for REVW

The central element of the REVW instruction is a three-cycle loop that is executed once for each word in the rule list. For the special case pass (where the \$FFE separator word is read between the rule antecedents and the rule consequents, and weights enabled by the C bit equal one), this loop takes five cycles. There is a small amount of housekeeping activity to get this loop started as REVW begins and a small sequence to end the instruction. If an interrupt comes, there is a special small sequence to save CPU status on the stack before the interrupt is serviced.

**Figure 9-10** is a detailed flow diagram for the REVW instruction. Each rectangular box represents one CPU clock cycle. Decision blocks and connecting arrows are considered to take no time at all. The letters in the small rectangles in the upper left corner of each bold box correspond to the execution cycle codes (refer to **SECTION 6 INSTRUCTION GLOSSARY** for details). Lower case letters indicate a cycle where 8-bit or no data is transferred. Upper case letters indicate cycles where 16-bit data could be transferred.

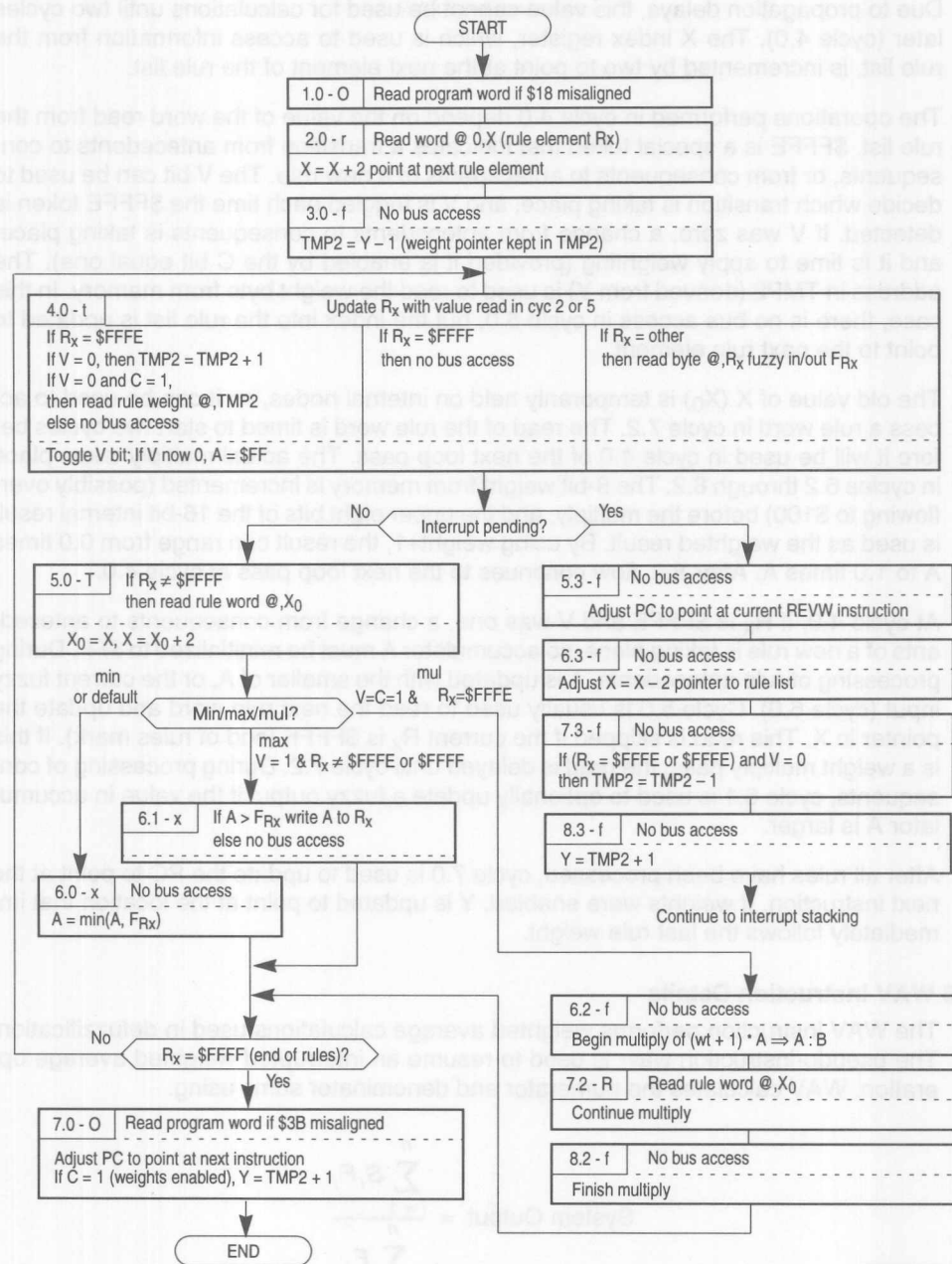


Figure 9-10 REVW Instruction Flow Diagram

In cycle 2.0, the first element of the rule list (a 16-bit address) is read from memory. Due to propagation delays, this value cannot be used for calculations until two cycles later (cycle 4.0). The X index register, which is used to access information from the rule list, is incremented by two to point at the next element of the rule list.

The operations performed in cycle 4.0 depend on the value of the word read from the rule list. \$FFFE is a special token that indicates a transition from antecedents to consequents, or from consequents to antecedents of a new rule. The V bit can be used to decide which transition is taking place, and V is toggled each time the \$FFFE token is detected. If V was zero, a change from antecedents to consequents is taking place, and it is time to apply weighting (provided it is enabled by the C bit equal one). The address in TMP2 (derived from Y) is used to read the weight byte from memory. In this case, there is no bus access in cycle 5.0, but the index into the rule list is updated to point to the next rule element.

The old value of X ( $X_0$ ) is temporarily held on internal nodes, so it can be used to access a rule word in cycle 7.2. The read of the rule word is timed to start two cycles before it will be used in cycle 4.0 of the next loop pass. The actual multiply takes place in cycles 6.2 through 8.2. The 8-bit weight from memory is incremented (possibly overflowing to \$100) before the multiply, and the upper eight bits of the 16-bit internal result is used as the weighted result. By using weight+1, the result can range from 0.0 times A to 1.0 times A. After 8.2, flow continues to the next loop pass at cycle 4.0.

At cycle 4.0, if  $R_x$  is \$FFFE and V was one, a change from consequents to antecedents of a new rule is taking place, so accumulator A must be reinitialized to \$FF. During processing of rule antecedents, A is updated with the smaller of A, or the current fuzzy input (cycle 6.0). Cycle 5.0 is usually used to read the next rule word and update the pointer in X. This read is skipped if the current  $R_x$  is \$FFFF (end of rules mark). If this is a weight multiply pass, the read is delayed until cycle 7.2. During processing of consequents, cycle 6.1 is used to optionally update a fuzzy output if the value in accumulator A is larger.

After all rules have been processed, cycle 7.0 is used to update the PC to point at the next instruction. If weights were enabled, Y is updated to point at the location that immediately follows the last rule weight.

## 9.6 WAV Instruction Details

The WAV instruction performs weighted average calculations used in defuzzification. The pseudo-instruction wavr is used to resume an interrupted weighted average operation. WAV calculates the numerator and denominator sums using:

$$\text{System Output} = \frac{\sum_{i=1}^n S_i F_i}{\sum_{i=1}^n F_i}$$



Where  $n$  is the number of labels of a system output,  $S_i$  are the singleton positions from the knowledge base, and  $F_i$  are fuzzy outputs from RAM.  $S_i$  and  $F_i$  are 8-bit values. The 8-bit B accumulator holds the iteration count  $n$ . Internal temporary registers hold intermediate sums, 24 bits for the numerator and 16 bits for the denominator. This makes this instruction suitable for  $n$  values up to 255 although eight is a more typical value. The final long division is performed with a separate EDIV instruction immediately after the WAV instruction. The WAV instruction returns the numerator and denominator sums in the correct registers for the EDIV. (EDIV performs the unsigned division  $Y = Y : D / X$ ; remainder in D).

Execution time for this instruction depends on the number of iterations (labels for the system output). WAV is interruptible so that worst case interrupt latency is not affected by the execution time for the complete weighted average operation. WAV includes initialization for the 24-bit and 16-bit partial sums so the first entry into WAV looks different than a resume from interrupt operation. The CPU12 handles this difficulty with a pseudo-instruction (wavr), which is specifically intended to resume an interrupted weighted average calculation. Refer to **9.6.3 Cycle-by-Cycle Details for WAV and wavr** for more detail.

#### 9.6.1 Setup Prior to Executing WAV

Before executing the WAV instruction, index registers X and Y and accumulator B must be set up. Index register X is a pointer to the  $S_i$  singleton list. X must have the address of the first singleton value in the knowledge base. Index register Y is a pointer to the fuzzy outputs  $F_i$ . Y must have the address of the first fuzzy output for this system output. B is the iteration count  $n$ . The B accumulator must be set to the number of labels for this system output.

#### 9.6.2 WAV Interrupt Details

The WAV instruction includes an 8-cycle processing loop for each label of the system output. Within this loop, the CPU checks whether a qualified interrupt request is pending. If an interrupt is detected, the current values of the internal temporary registers for the 24-bit and 16-bit sums are stacked, the CPU registers are stacked, and the interrupt is serviced.

A special processing sequence is executed when an interrupt is detected during a weighted average calculation. This exit sequence adjusts the PC so that it points to the second byte of the WAV object code (\$3C), before the PC is stacked. Upon return from the interrupt, the \$3C value is interpreted as a wavr pseudo-instruction. The wavr pseudo-instruction causes the CPU to execute a special WAV resumption sequence. The wavr recovery sequence adjusts the PC so that it looks like it did during execution of the original WAV instruction, then jumps back into the WAV processing loop. If another interrupt occurs before the weighted average calculation finishes, the PC is adjusted again as it was for the first interrupt. WAV can be interrupted any number of times, and additional WAV instructions can be executed while a WAV instruction is interrupted.

### 9.6.3 Cycle-by-Cycle Details for WAV and wavr

The WAV instruction is unusual in that the logic flow has two separate entry points. The first entry point is the normal start of a WAV instruction. The second entry point is used to resume the weighted average operation after a WAV instruction has been interrupted. This recovery operation is called the wavr pseudo-instruction.

**Figure 9-11** is a flow diagram of the WAV instruction including the wavr pseudo-instruction. Each rectangular box in this figure represents one CPU clock cycle. Decision blocks and connecting arrows are considered to take no time at all. The letters in the small rectangles in the upper left corner of the boxes correspond to execution cycle codes (refer to **SECTION 6 INSTRUCTION GLOSSARY** for details). Lower case letters indicate a cycle where 8-bit or no data is transferred. Upper case letters indicate cycles where 16-bit data could be transferred.

In terms of cycle-by-cycle bus activity, the \$18 page select prebyte is treated as a special 1-byte instruction. In cycle 1.0 of the WAV instruction, one word of program information will be fetched into the instruction queue if the \$18 is located at an odd address. If the \$18 is at an even address, the instruction queue cannot advance so there is no bus access in this cycle.

There is no bus access in cycles 2.0 or 3.0. In cycle 3.0, three internal 16-bit temporary registers are cleared in preparation for summation operations. The WAV instruction maintains a 32-bit sum-of-products in TMP3 : TMP2 and a 16-bit sum-of-weights in TMP1. By keeping these sums inside the CPU, bus accesses are reduced and the WAV operation is optimized for high speed.

Cycles 4.0 through 11.0 form the eight cycle main loop for WAV. The value in the 8-bit B accumulator is used to count the number of loop iterations. B is decremented at the top of the loop in cycle 4.0, and the test for zero is located at the bottom of the loop after cycle 11.0. Cycle 5.0 and 6.0 are used to fetch the 8-bit operands for one iteration of the loop. X and Y index registers are used to access these operands. The index registers are incremented as the operands are fetched. Cycle 7.0 is used to accumulate the current fuzzy output into TMP1. Cycles 8.0 through 10.0 are used to perform the eight by eight multiply of  $F_i$  times  $S_j$ . The multiply result is accumulated into TMP3 : TMP2 during cycles 10.0 and 11.0. Even though the sum-of-products will not exceed 24 bits, the sum is maintained in the 32-bit combined TMP3 : TMP2 register because it is easier to use existing 16-bit operations than it would be to create a new smaller operation to handle the high order bits of this sum.

Since the weighted average operation could be quite long, it is made to be interruptible. The usual longest latency path is from very early in cycle 7.0, through cycle 11.0, to the top of the loop to cycle 4.0, through cycle 6.0 to the interrupt check. There is also a three cycle (7.1 through 9.1) exit sequence making this latency path a total of 12 cycles. There is an even longer path, but it is much less likely to occur. If an interrupt comes near the beginning of cycle 2.1, when a weighted average operation is being resumed after a previous interrupt, the latency path is 2.1 through 6.1 plus 7.0 through 11.0 plus 4.0 through 6.0 plus the exit 7.1 through 9.1. This is a worst-case total of 17 cycles.

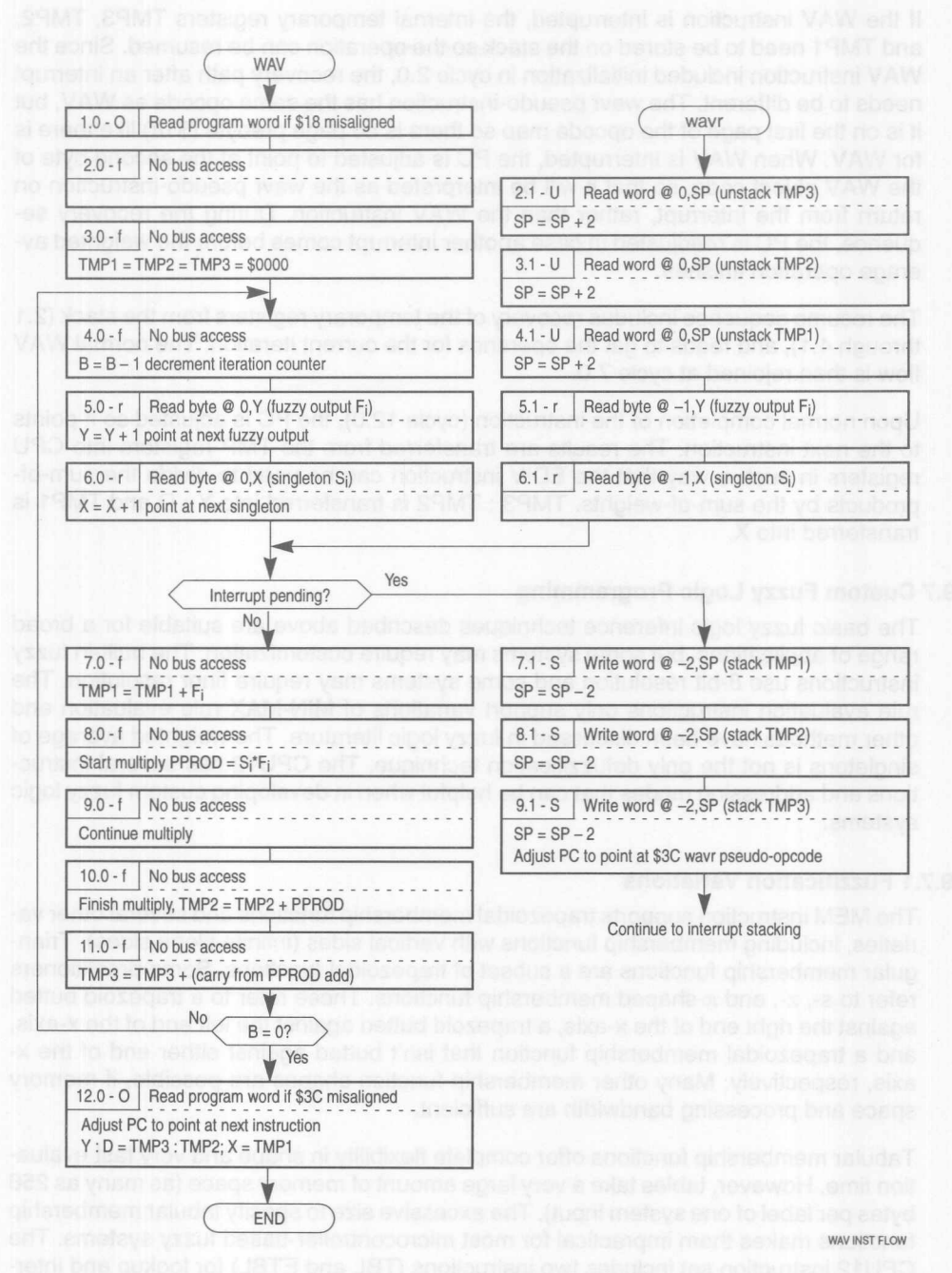


Figure 9-11 WAV and wavr Instruction Flow Diagram



If the WAV instruction is interrupted, the internal temporary registers TMP3, TMP2, and TMP1 need to be stored on the stack so the operation can be resumed. Since the WAV instruction included initialization in cycle 2.0, the recovery path after an interrupt needs to be different. The wavr pseudo-instruction has the same opcode as WAV, but it is on the first page of the opcode map so there is no page prebyte (\$18) like there is for WAV. When WAV is interrupted, the PC is adjusted to point at the second byte of the WAV object code, so that it will be interpreted as the wavr pseudo-instruction on return from the interrupt, rather than the WAV instruction. During the recovery sequence, the PC is readjusted in case another interrupt comes before the weighted average operation finishes.

The resume sequence includes recovery of the temporary registers from the stack (2.1 through 4.1), and reads to get the operands for the current iteration. The normal WAV flow is then rejoined at cycle 7.0.

Upon normal completion of the instruction (cycle 12.0), the PC is adjusted so it points to the next instruction. The results are transferred from the TMP registers into CPU registers in such a way that the EDIV instruction can be used to divide the sum-of-products by the sum-of-weights. TMP3 : TMP2 is transferred into Y : D and TMP1 is transferred into X.

## **9.7 Custom Fuzzy Logic Programming**

The basic fuzzy logic inference techniques described above are suitable for a broad range of applications, but some systems may require customization. The built-in fuzzy instructions use 8-bit resolution and some systems may require finer resolution. The rule evaluation instructions only support variations of MIN-MAX rule evaluation and other methods have been discussed in fuzzy logic literature. The weighted average of singletons is not the only defuzzification technique. The CPU12 has several instructions and addressing modes that can be helpful when in developing custom fuzzy logic systems.

### **9.7.1 Fuzzification Variations**

The MEM instruction supports trapezoidal membership functions and several other varieties, including membership functions with vertical sides (infinite slope sides). Triangular membership functions are a subset of trapezoidal functions. Some practitioners refer to s-, z-, and  $\pi$ -shaped membership functions. These refer to a trapezoid butted against the right end of the x-axis, a trapezoid butted against the left end of the x-axis, and a trapezoidal membership function that isn't butted against either end of the x-axis, respectively. Many other membership function shapes are possible, if memory space and processing bandwidth are sufficient.

Tabular membership functions offer complete flexibility in shape and very fast evaluation time. However, tables take a very large amount of memory space (as many as 256 bytes per label of one system input). The excessive size to specify tabular membership functions makes them impractical for most microcontroller-based fuzzy systems. The CPU12 instruction set includes two instructions (TBL and ETBL) for lookup and interpolation of compressed tables.

The TBL instruction uses 8-bit table entries (y-values) and returns an 8-bit result. The ETBL instruction uses 16-bit table entries (y-values) and returns a 16-bit result. A flexible indexed addressing mode is used to identify the effective address of the data point at the beginning of the line segment, and the data value for the end point of the line segment is the next consecutive memory location (byte for TBL and word for ETBL). In both cases, the B accumulator represents the ratio of (the x-distance from the beginning of the line segment to the lookup point) to (the x-distance from the beginning of the line segment to the end of the line segment). B is treated as an 8-bit binary fraction with radix point left of the MSB, so each line segment can effectively be divided into 256 pieces. During execution of the TBL or ETBL instruction, the difference between the end point y-value and the beginning point y-value (a signed byte-TBL or word-ETBL) is multiplied by the B accumulator to get an intermediate delta-y term. The result is the y-value of the beginning point, plus this signed intermediate delta-y value.

Because indexed addressing mode is used to identify the starting point of the line segment of interest, there is a great deal of flexibility in constructing tables. A common method is to break the x-axis range into 256 equal width segments and store the y value for each of the resulting 257 endpoints. The 16-bit D accumulator is then used as the x input to the table. The upper eight bits (A) is used as a coarse lookup to find the line segment of interest, and the lower eight bits (B) is used to interpolate within this line segment.

In the program sequence...

```
LDX      #TBL_START
LDD      DATA_IN
TBL      A,X
```

The notation A,X causes the TBL instruction to use the A<sup>th</sup> line segment in the table. The low-order half of D (B) is used by TBL to calculate the exact data value from this line segment. This type of table uses only 257 entries to approximate a table with 16 bits of resolution. This type of table has the disadvantage of equal width line segments, which means just as many points are needed to describe a flat portion of the desired function as are needed for the most active portions.

Another type of table stores x:y coordinate pairs for the endpoints of each linear segment. This type of table may reduce the table storage space compared to the previous fixed-width segments because flat areas of the functions can be specified with a single pair of endpoints. This type of table is a little harder to use with the CPU12 TBL and ETBL instructions because the table instructions expect y-values for segment endpoints to be in consecutive memory locations.

Consider a table made up of an arbitrary number of x:y coordinate pairs, where all values are eight bits. The table is entered with the x-coordinate of the desired point to lookup in the A accumulator. When the table is exited, the corresponding y-value is in the A accumulator. **Figure 9-12** shows one way to work with this type of table.

BEGIN	LDY	#TABLE_START-2	;setup initial table pointer
FIND_LOOP	CMPA	2,+Y	;find first Xn > XL
			; (auto pre-inc Y by 2)
	BLS	FIND_LOOP	;loop if XL .le. Xn
* on fall thru,	XB@-2,Y	YB@-1,Y	XE@0,Y and YE@1,Y
	TFR	D,X	;save XL in high half of X
	CLRA		;zero upper half of D
	LDAB	0,Y	;D = 0:XE
	SUBB	-2,Y	;D = 0:(XE-XB)
	EXG	D,X	;X = (XE-XB).. D = XL:junk
	SUBA	-2,Y	;A = (XL-XB)
	EXG	A,D	;D = 0:(XL-XB), uses trick of EXG
	FDIV		;X reg = (XL-XB)/(XE-XB)
	EXG	D,X	;move fractional result to A:B
	EXG	A,B	;byte swap - need result in B
	TSTA		;check for rounding
	BPL	NO_ROUND	
	INCB		;round B up by 1
NO_ROUND	LDAA	1,Y	;YE
	PSHA		;put on stack for TBL later
	LDAA	-1,Y	;YB
	PSHA		;now YB@0,SP and YE@1,SP
	TBL	2,SP+	;interpolate and deallocate
			;stack temps

**Figure 9-12 Endpoint Table Handling**

The basic idea is to find the segment of interest, temporarily build a one-segment table of the correct format on the stack, then use TBL with stack relative indexed addressing to interpolate. The most difficult part of the routine is calculating the proportional distance from the beginning of the segment to the lookup point versus the width of the segment  $((XL-XB)/(XE-XB))$ . With this type of table, this calculation must be done at run time. In the previous type of table, this proportional term is an inherent part (the lowest order bits) of the data input to the table.

Some fuzzy theorists have suggested membership functions should be shaped like normal distribution curves or other mathematical functions. This may be correct, but the processing requirements to solve for an intercept on such a function would be unacceptable for most microcontroller-based fuzzy systems. Such a function could be encoded into a table of one of the previously described types.

For many common systems, the thing that is most important about membership function shape is that there is a gradual transition from non-membership to membership as the system input value approaches the central range of the membership function. Let us examine the human problem of stopping a car at an intersection. We might use rules like "If intersection is close and speed is fast, apply brakes." The meaning (reflected in membership function shape and position) of the labels "close" and "fast" will be different for a teenager than they are for a grandmother, but both can accomplish the goal of stopping. It makes intuitive sense that the exact shape of a membership function is much less important than the fact that it has gradual boundaries.

### 9.7.2 Rule Evaluation Variations

The REV and REVW instructions expect fuzzy input and fuzzy output values to be 8-bit values. In a custom fuzzy inference program, higher resolution may be desirable (although this is not a common requirement). The CPU12 includes variations of minimum and maximum operations that work with the fuzzy MIN-MAX inference algorithm. The problem with the fuzzy inference algorithm is that the min and max operations need to store their results differently, so the min and max instructions must work differently or more than one variation of these instructions is needed.

The CPU12 has min and max instructions for 8- or 16-bit operands, where one operand is in an accumulator and the other is a referenced memory location. There are separate variations that replace the accumulator or the memory location with the result. While processing rule antecedents in a fuzzy inference program, a reference value must be compared to each of the referenced fuzzy inputs, and the smallest input must end up in an accumulator. The instruction...

```
EMIND      2,X+      ;process one rule antecedent
```

automates the central operations needed to process rule antecedents. The E stands for extended, so this instruction compares 16-bit operands. The D at the end of the mnemonic stands for the D accumulator, which is both the first operand for the comparison and the destination of the result. The 2,X+ is an indexed addressing specification that says X points to the second operand for the comparison.

When processing rule consequents, the operand in the accumulator must remain constant (in case there is more than one consequent in the rule), and the result of the comparison must replace the referenced fuzzy output in RAM. To do this, use the instruction...

```
EMAXM      2,X+      ;process one rule consequent
```

The M at the end of the mnemonic indicates that the result will replace the referenced memory operand. Again, indexed addressing is used. These two instructions would form the working part of a 16-bit resolution fuzzy inference routine.

There are many other methods of performing inference, but none of these are as widely used as the min-max method. Since the CPU12 is a general-purpose microcontroller, the programmer has complete freedom to program any algorithm desired. A custom programmed algorithm would typically take more code space and execution time than a routine that used the built-in REV or REVW instructions.

### 9.7.3 Defuzzification Variations

There are two main areas where other CPU12 instructions can help with custom defuzzification routines. The first case is working with operands that are more than eight bits. The second case involves using an entirely different approach than weighted average of singletons.

The primary part of the WAV instruction is a multiply and accumulate operation to get the numerator for the weighted average calculation. When working with operands as large as 16 bits, the EMACS instruction could at least be used to automate the multiply and accumulate function. The CPU12 has extended math capabilities, including the EMACS instruction which uses 16-bit input operands and accumulates the sum to a 32-bit memory location and 32-bit by 16-bit divide instructions.

One benefit of the WAV instruction is that both a sum of products and a sum of weights are maintained, while the fuzzy output operand is only accessed from memory once. Since memory access time is such a significant part of execution time, this provides a speed advantage compared to conventional instructions.

The weighted average of singletons is the most commonly used technique in micro-controllers because it is computationally less difficult than most other methods. The simplest method is called max defuzzification, which simply uses the largest fuzzy output as the system result. However, this approach does not take into account any other fuzzy outputs, even when they are almost as true as the chosen max output. Max defuzzification is not a good general choice because it only works for a subset of fuzzy logic applications.

The CPU12 is well suited for more computationally challenging algorithms than weighted average. A 32-bit by 16-bit divide instruction takes eleven or twelve 8-MHz cycles for unsigned or signed variations. A 16-bit by 16-bit multiply with a 32-bit result takes only three 8-MHz cycles. The EMACS instruction uses 16-bit operands and accumulates the result in a 32-bit memory location, taking only twelve 8-MHz cycles per iteration, including accessing all operands from memory and storing the result to memory.



## SECTION 10

### MEMORY EXPANSION

This section discusses expansion memory principles that apply to the entire M68HC12 family. Some family devices do not have memory expansion capabilities, and the size of the expanded memory can also vary. Please refer to the documentation for a derivative to determine details of implementation.

#### 10.1 Expansion System Description

Certain members of the M68HC12 family incorporate hardware that supports addressing a larger memory space than the standard 64 Kbytes. The expanded memory system uses fast on-chip logic to implement a transparent paged memory or bank-switching scheme.

Increased code efficiency is the greatest advantage of using bank switching instead of implementing a large linear address space. In systems with large linear address spaces, instructions require more bits of information to address a memory location, and CPU overhead is greater. Other advantages of bank switching include the ability to change the size of system memory, and the ability to use various types of external memory.

However, the add-on bank switching schemes used in other microcontrollers have known weaknesses. These include the cost of external glue logic, increased programming overhead to change banks, and the need to disable interrupts while banks are switched.

The M68HC12 system requires no external glue logic. Bank switching overhead is reduced by implementing control logic in the MCU. Interrupts do not need to be disabled during switching because switching tasks are incorporated in special instructions that greatly simplify program access to extended memory. Operation of the bank-switching logic is transparent to the CPU.

The CPU12 has a linear 64-Kbyte address space. All MCU system resources, including control registers for on-chip peripherals and on-chip memory arrays, are mapped into this space. In a typical M68HC12 derivative, the resources have default addresses out of reset, but can be re-mapped to other addresses by means of control registers in the on-chip integration module.

Memory expansion control logic is outside the CPU. A block of circuitry in the MCU integration module manages overlays that occupy pre-defined locations in the 64-Kbyte space addressed by the CPU. These overlays can be thought of as windows through which the CPU accesses information in the expanded memory space.

There are three overlay windows. The program window expands program memory, the data window is used for independent data expansion, and the extra window expands access to special types of memory such as EEPROM. The program window always occupies the 16-Kbyte space from \$8000 to \$BFFF. Data and extra windows can vary in size and location.

Each window has an associated page select register that selects external memory pages to be accessed via the window. Only one page at a time can occupy a window; the value in the register must be changed to access a different page of memory. With 8-bit registers, there can be up to 256 expansion pages per window, each page the same size as the window.

For data and extra windows, page switching is accomplished by means of normal read and write instructions. This is the traditional method of managing a bank-switching system. The CPU12 CALL and RTC instructions automatically manipulate the program page select (PPAGE) register for the program window.

In M68HC12 expanded memory systems, control registers, vector spaces, and a portion of on-chip memory are located in unpagged portions of the 64-Kbyte address space. The stack and I/O addresses should also be placed in unpagged memory to make them accessible from any overlay page.

The initial portions of exception handlers must be located in unpagged memory because the 16-bit exception vectors cannot point to addresses in paged memory. However, service routines can call other routines in paged memory. The upper 16-Kbyte block of memory space (\$C000–\$FFFF) is unpagged. It is recommended that all reset and interrupt vectors point to locations in this area.

Although internal MCU resources, such as control registers and on-chip memory, have default addresses out of reset, each can typically be relocated by changing the default values in control registers. Normally, I/O addresses, control registers, vector spaces, overlay windows, and on-chip memory are not mapped so that their respective address ranges overlap. However, there is an access priority order that prevents access conflicts should such overlaps occur. **Table 10-1** shows the mapping precedence. Resources with higher precedence block access to those with a lower precedence. The windows have lowest priority — registers, exception vectors, and on-chip memory are always visible to a program regardless of the values in the page select registers.

**Table 10-1 Mapping Precedence**

Precedence	Resource
1	Registers
2	Exception Vectors/BDM ROM
3	RAM
4	EEPROM
5	Flash
6	Expansion Windows

When background debugging is enabled and active, the CPU executes code located in a small on-chip ROM mapped to addresses \$FF20 to \$FFFF, and BDM control registers are accessible at addresses \$FF00 to \$FF06. The BDM ROM replaces the regular system vectors while BDM is active, but BDM resources are not in the memory map during normal execution of application programs.

## 10.2 CALL and Return from Call Instructions

The CALL is similar to a JSR instruction, but the subroutine that is called can be located anywhere in the normal 64-Kbyte address space, or on any page of program expansion memory. When CALL is executed, a return address is calculated, then it and the current program page register value are stacked, and a new instruction-supplied value is written to PPAGE. The PPAGE value controls which of the 256 possible pages is visible through the 16-Kbyte window in the 64-Kbyte memory map. Execution continues at the address of the called subroutine.

The actual sequence of operations that occur during execution of CALL is:

- The CPU reads the old PPAGE value into an internal temporary register, and writes the new instruction-supplied PPAGE value to PPAGE. This switches the destination page into the program overlay window.
- The CPU calculates the address of the next instruction after the CALL instruction (the return address), and pushes this 16-bit value onto the stack.
- The old 8-bit PPAGE value is pushed onto the stack.
- The effective address of the subroutine is calculated, the queue is refilled, and execution begins at the new address.

This sequence of operations is an uninterruptable CPU instruction. There is no need to inhibit interrupts during CALL execution. In addition, a CALL can be performed from any address in memory to any other address. This is a big improvement over other bank-switching schemes, where the page switch operation can only be performed by a program outside the overlay window.

For all practical purposes, the PPAGE value supplied by the instruction can be considered to be part of the effective address. For all addressing mode variations except indexed indirect modes, the new page value is provided by an immediate operand in the instruction. For indexed indirect variations of CALL, a pointer specifies memory locations where the new page value and the address of the called subroutine are stored. Use of indirect addressing for both the new page value and the address within the page allows use run-time calculated values rather than immediate values that must be known at the time of assembly.

The RTC instruction is used to terminate subroutines invoked by a CALL instruction. RTC unstacks the PPAGE value and the return address, the queue is refilled, and execution resumes with the next instruction after the corresponding CALL.

The actual sequence of operations that occur during execution of RTC is:

- The return value of the 8-bit PPAGE register is pulled from the stack.
- The 16-bit return address is pulled from the stack and loaded into the PC.
- The return PPAGE value is written to the PPAGE register.
- The queue is refilled, and execution begins at the new address.

Since the return operation is implemented as a single uninterruptable CPU instruction, the RTC can be executed from anywhere in memory, including from a different page of extended memory in the overlay window.



In an MCU where there is no memory expansion, the CALL and RTC instructions still perform the same sequence of operations, but there is no PPAGE register or address translation logic. The value the CPU reads when the PPAGE register is accessed is indeterminate but doesn't matter, because the value is not involved in addressing memory in the unpaged 64-Kbyte memory map. When the CPU writes to the non-existent PPAGE register, nothing happens.

The CALL and RTC instructions behave like JSR and RTS, except they have slightly longer execution times. Since extra execution cycles are required, routinely substituting CALL/RTC for JSR/RTS is not recommended. JSR and RTS can be used to access subroutines that are located on the same memory page. However, if a subroutine can be called from other pages, it must be terminated with an RTC. In this case, since RTC unstacks the PPAGE value as well as the return address, all accesses to the subroutine, even those made from the same page, must use CALL instructions.

### 10.3 Address Lines for Expansion Memory

All M68HC12 family members have at least 16 address lines, ADDR[15:0]. Devices with memory expansion capability can have as many as six additional high-order external address lines, ADDR[21:16]. Each of these additional address lines is typically associated with a control bit that allows address expansion to be selectively enabled. When expansion is enabled, internal address translation circuitry multiplexes data from the page select registers onto the high order address lines when there is an access to an address in a corresponding expansion window.

Assume that a device has six expansion address lines and an 8-bit PPAGE register. The lines and the program expansion window have been enabled. The address \$9000 is within the 16-Kbyte program overlay window. When there is an access to this address, the value in the PPAGE register is multiplexed onto external address lines ADDR[21:14]. The 14 low-order address lines select a location within the program overlay page. Up to 256 16-Kbyte pages (4 Mbytes) of memory can be accessed through the window. When there is an access to a location that is not within any enabled overlay window, ADDR[21:16] are driven to logic level one.

The address translation logic can produce the same address on the external address lines for two different internal addresses. For example, the 22-bit address \$3FFFFFF could result from an internal access to \$FFFF in the 64-Kbyte memory map, or to the last location (\$BFFF) within page 255 (PPAGE = \$FF) of the program overlay window. Considering only the 22 external address lines, the last physical page of the program overlay appears to occupy the same address space as the unpaged 16-Kbyte block from \$C000 to \$FFFF of the 64-Kbyte memory map. Using MCU chip-select circuits to enable external memory can resolve these ambiguities.

### 10.4 Overlay Window Controls

There is a page select register associated with each overlay window. PPAGE holds the page select for the program overlay, DPAGE holds the page select for the data overlay, and EPAGE holds the page select for the extra page. The CPU12 manipulates the PPAGE register directly, so it will always be eight bits or less in devices that

support program memory expansion. The DPAGE and EPAGE registers are not controlled by dedicated CPU12 instructions. These registers can be larger or smaller than eight bits in various M68HC12 derivatives.

Typically, each of the overlay windows also has an associated control bit to enable memory expansion through the appropriate window. Memory expansion is generally disabled out of reset, so control bits must be written to enable the address translation logic.

### **10.5 Using Chip-Select Circuits**

M68HC12 chip-select circuits can be used to preclude ambiguities in memory-mapping due to the operation of internal address translation logic. If built-in chip selects are not used, take care to use only overlay pages which produce unique addresses on the external address lines.

M68HC12 derivatives typically have two or more chip-select circuits. Chip-select function is conceptually simple. Whenever an access to a pre-defined range of addresses is made, internal MCU circuitry detects an address match and asserts a control signal that can be used to enable external devices. Chip-select circuits typically incorporate a number of options that make it possible to use more than one range of addresses for matches as well as to enable various types and configurations of external devices.

Chip-select circuits used in conjunction with the memory-expansion scheme must be able to match all accesses made to addresses within the appropriate program overlay window. In the case of the program expansion window, the range of addresses occupies the 16-Kbyte space from \$8000 to \$BFFF. For data and extra expansion windows, the range of addresses varies from device to device. The following paragraphs discuss a typical implementation of memory expansion chip-select functions in the system integration module. Implementation will vary from device to device within the M68HC12 family. Please refer to the appropriate device manual for details.

#### **10.5.1 Program Memory Expansion Chip-Select Controls**

There are two program memory expansion chip-select circuits, CSP0 and CSP1. The associated control register contains eight control bits that provide for a number of system configurations.

##### **10.5.1.1 CSP1E Control Bit**

Enables (1) or disables (0) the CSP1 chip select. The default is disabled.

##### **10.5.1.2 CSP0E Control Bit**

Enables (1) or disables (0) the CSP0 chip select. The default is enabled. This allows CSP0 to be used to select an external memory that includes the reset vector and start-up initialization programs.

#### 10.5.1.3 CSP1FL Control Bit

Configures CSP1 to occupy all of the 64-Kbyte memory map that is not used by a higher-priority resource. If CSP1FL = 0, CSP1 is mapped to the area from \$8000 to \$FFFF. CSP1 has the lowest access priority except for external memory space that is not associated with any chip select.

#### 10.5.1.4 CSPA21 Control Bit

Logic one causes CSP0 and CSP1 to be controlled by the ADDR21 signal. CSP1 is active when ADDR21 = 0, and CSP0 is active when ADDR21 = 1. When CSPA21 is one, the CSP1FL bit is ignored and both CSP0 and CSP1 are active in the region \$8000–\$FFFF. When CSPA21 is zero, CSP0 and CSP1 operate independently from the value of the ADDR21 signal.

#### 10.5.1.5 STRP0A:STRP0B Control Field

These two bits program an extra delay into accesses to the CSP0 area of memory. The choices are 0, 1, 2, or 3 E-cycles in addition to the normal one cycle for unstretched accesses. This allows use of slow external memory without slowing down the entire system.

#### 10.5.1.6 STRP1A:STRP1B Control Field

These two bits program an extra delay into accesses to the CSP1 area of memory. The choices are 0, 1, 2, or 3 E-cycles in addition to the normal one cycle for unstretched accesses. This allows use of slow external memory without slowing down the entire system.

When enabled, CSP0 is active for the memory space from \$8000 through \$FFFF. This includes the program overlay space (\$8000–\$BFFF) and the unpaged 16-Kbyte block from \$C000 through \$FFFF. This configuration can be used if there is a single program memory device (up to four Mbytes) in the system.

If CSP1 is also enabled and the CSPA21 bit is set, CSP1 can be used to select the first 128 16-Kbyte pages (two Mbytes) in the program overlay expansion memory space while CSP0 selects the higher numbered program expansion pages and the unpaged block from \$C000 through \$FFFF. Recall that the external memory device cannot distinguish between an access to the \$C000 to \$FFFF space and an access to \$8000–\$BFFF in the 255th page (PPAGE = \$FF) of the program overlay window.

### 10.5.2 Data Expansion Chip Select Controls

The data chip select (CSD) has four associated control bits.

#### 10.5.2.1 CSDE Control Bit

Enables (1) or disables (0) the CSD chip select. The default is disabled.

#### **10.5.2.2 CSDHF Control Bit**

Configures CSD to occupy the lower half of the 64-Kbyte memory map (for areas that are not used by a higher priority resource). If CSDHF is zero, CSD occupies the range of addresses used by the data expansion window.

#### **10.5.2.3 STRDA:STRDB Control Field**

These two bits program an extra delay into accesses to the CSD area of memory. The choices are 0, 1, 2, or 3 additional E-cycles in addition to the normal one cycle for unstretched accesses. This allows use of slow external memory without slowing down the entire system.

### **10.5.3 Extra Expansion Chip Select Controls**

The extra chip select (CSE) has four associated control bits.

#### **10.5.3.1 CSEE Control Bit**

Enables (1) or disables (0) the CSE chip select. The default is disabled.

#### **10.5.3.2 CSEEP Control Bit**

Logic one configures CSE to be active for the EPAGE area. A logic zero causes CSE to be active for the CS3 area of the internal register space, which can typically be remapped to any 2-Kbyte boundary.

#### **10.5.3.3 STREA:STREB Control Field**

These two bits program an extra delay into accesses to the CSE area of memory. The choices are 0, 1, 2, or 3 E-cycles in addition to the normal one cycle for unstretched accesses. This allows use of slow external memory without slowing down the entire system.

To use CSE with the extra overlay window, it must be enabled (CSEE = 1) and configured to follow the extra page (CSEEP = 1).

### **10.6 System Notes**

The expansion overlay windows are specialized for specific application uses, but there are no restrictions on the use of these memory spaces. Motorola MCUs have a memory-mapped architecture in which all memory resources are treated equally. Although it is possible to execute programs in paged external memory in the data and extra overlay areas, it is less convenient than using the program overlay area.

The CALL and RTC instructions automate the program page switching functions in an uninterruptable instruction. For the data and extra overlay windows, the user must take care not to let interrupts corrupt the page switching sequence or change the active page while executing out of another page in the same overlay area.

Internal MCU chip-select circuits have access to all 16 internal CPU address lines and the overlay window select lines. This allows all 256 expansion pages in an overlay window to be distinguished from unpaged memory locations with 22-bit addresses that are the same as addresses in overlay pages.

**10.5.2.2 CSDBF Control Bit**

Configures CSB to occupy the lower half of the 64-Kbyte memory map (for areas that are not used by a higher priority resource). If CSDBF is zero, CSB occupies the range of addresses used by the data expansion window.

**10.5.2.3 STRDA:STRDB Control Field**

These two bits program an extra delay into accesses to the CSB area of memory. The choices are 0, 1, 2, or 3 additional E-cycles in addition to the normal one cycle for unattached accesses. This allows use of slow external memory without slowing down the entire system.

**10.5.2.4 Extra Expansion Chip Select Controls**

The extra chip select (CSE) has four associated control bits.

**10.5.2.4.1 CSE Control Bit**

Enables (1) or disables (0) the CSE chip select. The default is disabled.

**10.5.2.4.2 CSBEP Control Bit**

Logic one configures CSE to be active for the EPAGE area. A logic zero causes CSE to be active for the CSB area of the internal register space, which can typically be mapped to any 2-Kbyte boundary.

**10.5.2.4.3 STRSA:STRSB Control Field**

These two bits program an extra delay into accesses to the CSE area of memory. The choices are 0, 1, 2, or 3 E-cycles in addition to the normal one cycle for unattached accesses. This allows use of slow external memory without slowing down the entire system.

To use CSE with the extra overlay window, it must be enabled (CSEB = 1) and configured to follow the extra page (CSBEP = 1).

## 10.6 System Notes

The expansion overlay windows are specialized for specific application uses, but there are no restrictions on the use of these memory spaces. Motorola MCUs have a memory-mapped architecture in which all memory resources are treated equally. Although it is possible to execute programs in paged external memory in the data and extra overlay areas, it is less convenient than using the program overlay area.

The CALL and RJC instructions automate the program page switching function in an unambiguous instruction. For the data and extra overlay windows, the user must take care not to let interrupts corrupt the page switching sequence or change the active page while executing out of another page in the same overlay area.

Internal MCU chip-select circuits have access to all 16 internal CPU address lines and the overlay window select lines. This allows all 256 expansion pages in an overlay window to be distinguished from unmapped memory locations with 32-bit addresses that are the same as addresses in overlay pages.

## APPENDIX A INSTRUCTION REFERENCE

### A.1 Instruction Set Summary

**Table A-1** is a quick reference to the CPU12 instruction set. The table shows source form, describes the operation performed, lists the addressing modes used, gives machine encoding in hexadecimal form, and describes the effect of execution on the condition code bits.

### A.2 Opcode Map

**Table A-2** displays the mnemonic, opcode, addressing mode, and cycle count for each instruction. The first table represents those opcodes with no prebyte. The second page of the table represents those opcodes with a prebyte value of \$18. Notice the first hexadecimal digit of the opcode (shown in the upper left corner of each cell) corresponds to column location, while the second hexadecimal digit of the opcode corresponds to row location.

### A.3 Indexed Addressing Postbyte Encoding

**Table A-5** shows postbyte encoding for indexed addressing modes. The mnemonic for the indexed addressing mode postbyte is xb. This is also the notation used in instruction glossary entries. **Table A-3** presents the same information in two-digit hexadecimal format. The first digit of the postbyte is represented by the value of the columns in the table. The second digit of the postbyte is represented by the value of the row.

### A.4 Transfer and Exchange Postbyte Encoding

**Table A-4** shows postbyte encoding for transfer and exchange instructions. The mnemonic for the transfer and exchange postbyte is eb. This is also the notation used in instruction glossary entries. The first digit of the instruction postbyte is related to the columns of the table. The second digit of the postbyte is related to the rows. The body of the table shows actions caused by the postbyte.

### A.5 Loop Primitive Postbyte Encoding

**Table A-6** shows postbyte encoding for loop primitive instructions. The mnemonic for the loop primitive postbyte is lb. This is also the notation used in instruction glossary entries. The loop primitive instructions are DBEQ, DBNE, IBEQ, IBNE, TBEQ, and TBNE. The first digit of the instruction postbyte corresponds to the columns of the table. The second digit of the postbyte corresponds to the rows. The body of the table shows actions caused by the postbyte.



**Table A-1 Instruction Set Summary**

Source Form	Operation	Addr. Mode	Machine Coding (hex)	~*	S	X	H	I	N	Z	V	C
ABA	(A) + (B) $\Rightarrow$ A Add Accumulators A and B	INH	18 06	2	-	-	$\Delta$	-	$\Delta$	$\Delta$	$\Delta$	$\Delta$
ABX	(B) + (X) $\Rightarrow$ X <i>Translates to LEAX B,X</i>	IDX	1A E5	2	-	-	-	-	-	-	-	-
ABY	(B) + (Y) $\Rightarrow$ Y <i>Translates to LEAY B,Y</i>	IDX	19 ED	2	-	-	-	-	-	-	-	-
ADCA <i>opr</i>	(A) + (M) + C $\Rightarrow$ A Add with Carry to A	IMM	89 ii	1	-	-	$\Delta$	-	$\Delta$	$\Delta$	$\Delta$	$\Delta$
		DIR	99 dd	3								
		EXT	B9 hh ll	3								
		IDX	A9 xb	3								
		IDX1	A9 xb ff	3								
		IDX2	A9 xb ee ff	4								
		[D,IDX] [IDX2]	A9 xb A9 xb ee ff	6 6								
ADCB <i>opr</i>	(B) + (M) + C $\Rightarrow$ B Add with Carry to B	IMM	C9 ii	1	-	-	$\Delta$	-	$\Delta$	$\Delta$	$\Delta$	$\Delta$
		DIR	D9 dd	3								
		EXT	F9 hh ll	3								
		IDX	E9 xb	3								
		IDX1	E9 xb ff	3								
		IDX2	E9 xb ee ff	4								
		[D,IDX] [IDX2]	E9 xb E9 xb ee ff	6 6								
ADDA <i>opr</i>	(A) + (M) $\Rightarrow$ A Add without Carry to A	IMM	8B ii	1	-	-	$\Delta$	-	$\Delta$	$\Delta$	$\Delta$	$\Delta$
		DIR	9B dd	3								
		EXT	BB hh ll	3								
		IDX	AB xb	3								
		IDX1	AB xb ff	3								
		IDX2	AB xb ee ff	4								
		[D,IDX] [IDX2]	AB xb AB xb ee ff	6 6								
ADDB <i>opr</i>	(B) + (M) $\Rightarrow$ B Add without Carry to B	IMM	CB ii	1	-	-	$\Delta$	-	$\Delta$	$\Delta$	$\Delta$	$\Delta$
		DIR	DB dd	3								
		EXT	FB hh ll	3								
		IDX	EB xb	3								
		IDX1	EB xb ff	3								
		IDX2	EB xb ee ff	4								
		[D,IDX] [IDX2]	EB xb EB xb ee ff	6 6								
ADDD <i>opr</i>	(A:B) + (M:M+1) $\Rightarrow$ A:B Add 16-Bit to D (A:B)	IMM	C3 jj kk	2	-	-	-	-	$\Delta$	$\Delta$	$\Delta$	$\Delta$
		DIR	D3 dd	3								
		EXT	F3 hh ll	3								
		IDX	E3 xb	3								
		IDX1	E3 xb ff	3								
		IDX2	E3 xb ee ff	4								
		[D,IDX] [IDX2]	E3 xb E3 xb ee ff	6 6								

Table A-1 Instruction Set Summary (Continued)

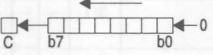
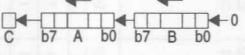
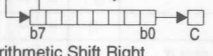
Source Form	Operation	Addr. Mode	Machine Coding (hex)	~*	S	X	H	I	N	Z	V	C
ANDA <i>opr</i>	(A) • (M) ⇒ A Logical And A with Memory	IMM DIR EXT IDX IDX1 IDX2 [D,IDX] [IDX2]	84 ii 94 dd B4 hh ll A4 xb A4 xb ff A4 xb ee ff A4 xb A4 xb ee ff	1 3 3 3 3 4 6 6	—	—	—	—	Δ	Δ	0	—
ANDB <i>opr</i>	(B) • (M) ⇒ B Logical And B with Memory	IMM DIR EXT IDX IDX1 IDX2 [D,IDX] [IDX2]	C4 ii D4 dd F4 hh ll E4 xb E4 xb ff E4 xb ee ff E4 xb E4 xb ee ff	1 3 3 3 3 4 6 6	—	—	—	—	Δ	Δ	0	—
ANDCC <i>opr</i>	(CCR) • (M) ⇒ CCR Logical And CCR with Memory	IMM	10 ii	1	↓	↓	↓	↓	↓	↓	↓	↓
ASL <i>opr</i>	 Arithmetic Shift Left	EXT IDX IDX1 IDX2 [D,IDX] [IDX2]	78 hh ll 68 xb 68 xb ff 68 xb ee ff 68 xb 68 xb ee ff	4 3 4 5 6 6	—	—	—	—	Δ	Δ	Δ	Δ
ASLA	Arithmetic Shift Left Accumulator A	INH	48	1								
ASLB	Arithmetic Shift Left Accumulator B	INH	58	1								
ASLD	 Arithmetic Shift Left Double	INH	59	1	—	—	—	—	Δ	Δ	Δ	Δ
ASR <i>opr</i>	 Arithmetic Shift Right	EXT IDX IDX1 IDX2 [D,IDX] [IDX2]	77 hh ll 67 xb 67 xb ff 67 xb ee ff 67 xb 67 xb ee ff	4 3 4 5 6 6	—	—	—	—	Δ	Δ	Δ	Δ
ASRA	Arithmetic Shift Right Accumulator A	INH	47	1								
ASRB	Arithmetic Shift Right Accumulator B	INH	57	1								
BCC <i>rel</i>	Branch if Carry Clear (if C = 0)	REL	24 rr	3/1	—	—	—	—	—	—	—	—
BCLR <i>opr, msk</i>	(M) • (mm) ⇒ M Clear Bit(s) in Memory	DIR EXT IDX IDX1 IDX2	4D dd mm 1D hh ll mm 0D xb mm 0D xb ff mm 0D xb ee ff mm	4 4 4 4 6	—	—	—	—	Δ	Δ	0	—
BCS <i>rel</i>	Branch if Carry Set (if C = 1)	REL	25 rr	3/1	—	—	—	—	—	—	—	—
BEQ <i>rel</i>	Branch if Equal (if Z = 1)	REL	27 rr	3/1	—	—	—	—	—	—	—	—
BGE <i>rel</i>	Branch if Greater Than or Equal (if N ⊕ V = 0) (signed)	REL	2C rr	3/1	—	—	—	—	—	—	—	—
BGND	Place CPU in Background Mode see Background Mode section.	INH	00	5	—	—	—	—	—	—	—	—
BGT <i>rel</i>	Branch if Greater Than (if Z + (N ⊕ V) = 0) (signed)	REL	2E rr	3/1	—	—	—	—	—	—	—	—
BHI <i>rel</i>	Branch if Higher (if C + Z = 0) (unsigned)	REL	22 rr	3/1	—	—	—	—	—	—	—	—



Table A-1 Instruction Set Summary (Continued)

Source Form	Operation	Addr. Mode	Machine Coding (hex)	~*	S	X	H	I	N	Z	V	C
BHS <i>rel</i>	Branch if Higher or Same (if C = 0) (unsigned) same function as BCC	REL	24 rr	3/1	-	-	-	-	-	-	-	-
BITA <i>opr</i>	(A) • (M) Logical And A with Memory	IMM	85 ii	1	-	-	-	-	Δ	Δ	0	-
		DIR	95 dd	3								
		EXT	B5 hh ll	3								
		IDX	A5 xb	3								
		IDX1	A5 xb ff	3								
		IDX2	A5 xb ee ff	4								
		[D,IDX] [IDX2]	A5 xb A5 xb ee ff	6 6								
BITB <i>opr</i>	(B) • (M) Logical And B with Memory	IMM	C5 ii	1	-	-	-	-	Δ	Δ	0	-
		DIR	D5 dd	3								
		EXT	F5 hh ll	3								
		IDX	E5 xb	3								
		IDX1	E5 xb ff	3								
		IDX2	E5 xb ee ff	4								
		[D,IDX] [IDX2]	E5 xb E5 xb ee ff	6 6								
BLE <i>rel</i>	Branch if Less Than or Equal (if Z + (N ⊕ V) = 1) (signed)	REL	2F rr	3/1	-	-	-	-	-	-	-	-
BLO <i>rel</i>	Branch if Lower (if C = 1) (unsigned) same function as BCS	REL	25 rr	3/1	-	-	-	-	-	-	-	-
BLS <i>rel</i>	Branch if Lower or Same (if C + Z = 1) (unsigned)	REL	23 rr	3/1	-	-	-	-	-	-	-	-
BLT <i>rel</i>	Branch if Less Than (if N ⊕ V = 1) (signed)	REL	2D rr	3/1	-	-	-	-	-	-	-	-
BMI <i>rel</i>	Branch if Minus (if N = 1)	REL	2B rr	3/1	-	-	-	-	-	-	-	-
BNE <i>rel</i>	Branch if Not Equal (if Z = 0)	REL	26 rr	3/1	-	-	-	-	-	-	-	-
BPL <i>rel</i>	Branch if Plus (if N = 0)	REL	2A rr	3/1	-	-	-	-	-	-	-	-
BRA <i>rel</i>	Branch Always (if 1 = 1)	REL	20 rr	3	-	-	-	-	-	-	-	-
BRCLR <i>opr, msk, rel</i>	Branch if (M) • (mm) = 0 (if All Selected Bit(s) Clear)	DIR	4F dd mm rr	4	-	-	-	-	-	-	-	-
		EXT	1F hh ll mm rr	5								
		IDX	0F xb mm rr	4								
		IDX1	0F xb ff mm rr	6								
		IDX2	0F xb ee ff mm rr	8								
BRN <i>rel</i>	Branch Never (if 1 = 0)	REL	21 rr	1	-	-	-	-	-	-	-	-
BRSET <i>opr, msk, rel</i>	Branch if (M̄) • (mm) = 0 (if All Selected Bit(s) Set)	DIR	4E dd mm rr	4	-	-	-	-	-	-	-	-
		EXT	1E hh ll mm rr	5								
		IDX	0E xb mm rr	4								
		IDX1	0E xb ff mm rr	6								
		IDX2	0E xb ee ff mm rr	8								
BSET <i>opr, msk</i>	(M) + (mm) ⇒ M Set Bit(s) in Memory	DIR	4C dd mm	4	-	-	-	-	Δ	Δ	0	-
		EXT	1C hh ll mm	4								
		IDX	0C xb mm	4								
		IDX1	0C xb ff mm	4								
		IDX2	0C xb ee ff mm	6								
BSR <i>rel</i>	(SP) - 2 ⇒ SP; RTN <sub>H</sub> :RTN <sub>L</sub> ⇒ M <sub>(SP)</sub> :M <sub>(SP+1)</sub> Subroutine address ⇒ PC	REL	07 rr	4	-	-	-	-	-	-	-	-
	Branch to Subroutine											

Table A-1 Instruction Set Summary (Continued)

Source Form	Operation	Addr. Mode	Machine Coding (hex)	~*	S	X	H	I	N	Z	V	C
BVC <i>rel</i>	Branch if Overflow Bit Clear (if V = 0)	REL	28 rr	3/1	-	-	-	-	-	-	-	-
BVS <i>rel</i>	Branch if Overflow Bit Set (if V = 1)	REL	29 rr	3/1	-	-	-	-	-	-	-	-
CALL <i>opr, page</i>	(SP) - 2 ⇒ SP; RTN <sub>H</sub> :RTN <sub>L</sub> ⇒ M <sub>(SP)</sub> :M <sub>(SP+1)</sub> (SP) - 1 ⇒ SP; (PPG) ⇒ M <sub>(SP)</sub> ; pg ⇒ PPAGE register; Program address ⇒ PC  Call subroutine in extended memory (Program may be located on another expansion memory page.)	EXT IDX IDX1 IDX2	4A hh ll pg 4B xb pg 4B xb ff pg 4B xb ee ff pg	8 8 8 9	-	-	-	-	-	-	-	-
CALL [D,r] CALL { <i>opr,r</i> }	Indirect modes get program address and new pg value based on pointer.  <i>r</i> = X, Y, SP, or PC	[D,IDX] [IDX2]	4B xb 4B xb ee ff	10 10	-	-	-	-	-	-	-	-
CBA	(A) - (B) Compare 8-Bit Accumulators	INH	18 17	2	-	-	-	-	Δ	Δ	Δ	Δ
CLC	0 ⇒ C <i>Translates to ANDCC #\$FE</i>	IMM	10 FE	1	-	-	-	-	-	-	-	0
CLI	0 ⇒ I <i>Translates to ANDCC #\$EF</i> (enables I-bit interrupts)	IMM	10 EF	1	-	-	-	0	-	-	-	-
CLR <i>opr</i>	0 ⇒ M Clear Memory Location	EXT IDX IDX1 IDX2 [D,IDX] [IDX2]	79 hh ll 69 xb 69 xb ff 69 xb ee ff 69 xb 69 xb ee ff	3 2 3 3 5 5	-	-	-	-	0	1	0	0
CLRA	0 ⇒ A Clear Accumulator A	INH	87	1	-	-	-	-	-	-	-	-
CLRB	0 ⇒ B Clear Accumulator B	INH	C7	1	-	-	-	-	-	-	-	-
CLV	0 ⇒ V <i>Translates to ANDCC #\$FD</i>	IMM	10 FD	1	-	-	-	-	-	-	0	-
CMPA <i>opr</i>	(A) - (M) Compare Accumulator A with Memory	IMM DIR EXT IDX IDX1 IDX2 [D,IDX] [IDX2]	81 ii 91 dd B1 hh ll A1 xb A1 xb ff A1 xb ee ff A1 xb A1 xb ee ff	1 3 3 3 3 4 6 6	-	-	-	-	Δ	Δ	Δ	Δ
CMPB <i>opr</i>	(B) - (M) Compare Accumulator B with Memory	IMM DIR EXT IDX IDX1 IDX2 [D,IDX] [IDX2]	C1 ii D1 dd F1 hh ll E1 xb E1 xb ff E1 xb ee ff E1 xb E1 xb ee ff	1 3 3 3 3 4 6 6	-	-	-	-	Δ	Δ	Δ	Δ

Table A-1 Instruction Set Summary (Continued)

Source Form	Operation	Addr. Mode	Machine Coding (hex)	~*	S	X	H	I	N	Z	V	C
COM <i>opr</i>	$(\bar{M}) \Rightarrow M$ equivalent to $\$FF - (M) \Rightarrow M$ 1's Complement Memory Location	EXT	71 hh ll	4	-	-	-	-	$\Delta$	$\Delta$	0	1
		IDX	61 xb	3								
		IDX1	61 xb ff	4								
		IDX2	61 xb ee ff	5								
		[D,IDX]	61 xb	6								
		[IDX2]	61 xb ee ff	6								
COMA	$(\bar{A}) \Rightarrow A$ Complement Accumulator A	INH	41	1								
COMB	$(\bar{B}) \Rightarrow B$ Complement Accumulator B	INH	51	1								
CPD <i>opr</i>	$(A:B) - (M:M+1)$ Compare D to Memory (16-Bit)	IMM	8C jj kk	2	-	-	-	-	$\Delta$	$\Delta$	$\Delta$	$\Delta$
		DIR	9C dd	3								
		EXT	BC hh ll	3								
		IDX	AC xb	3								
		IDX1	AC xb ff	3								
		IDX2	AC xb ee ff	4								
CPS <i>opr</i>	$(SP) - (M:M+1)$ Compare SP to Memory (16-Bit)	[D,IDX]	AC xb	6								
		[IDX2]	AC xb ee ff	6								
		IMM	8F jj kk	2	-	-	-	-	$\Delta$	$\Delta$	$\Delta$	$\Delta$
		DIR	9F dd	3								
		EXT	BF hh ll	3								
		IDX	AF xb	3								
CPX <i>opr</i>	$(X) - (M:M+1)$ Compare X to Memory (16-Bit)	IDX1	AF xb ff	3								
		IDX2	AF xb ee ff	4								
		[D,IDX]	AF xb	6								
		[IDX2]	AF xb ee ff	6								
		IMM	8E jj kk	2	-	-	-	-	$\Delta$	$\Delta$	$\Delta$	$\Delta$
		DIR	9E dd	3								
CPY <i>opr</i>	$(Y) - (M:M+1)$ Compare Y to Memory (16-Bit)	EXT	BE hh ll	3								
		IDX	AE xb	3								
		IDX1	AE xb ff	3								
		IDX2	AE xb ee ff	4								
		[D,IDX]	AE xb	6								
		[IDX2]	AE xb ee ff	6								
DAA	Adjust Sum to BCD Decimal Adjust Accumulator A	INH	18 07	3	-	-	-	-	$\Delta$	$\Delta$	?	$\Delta$
DBEQ <i>cntr, rel</i>	$(cntr) - 1 \Rightarrow cntr$ if $(cntr) = 0$ , then Branch else Continue to next instruction  Decrement Counter and Branch if = 0 ( $cntr = A, B, D, X, Y$ , or SP)	REL (9-bit)	04 lb rr	3	-	-	-	-	-	-	-	-
DBNE <i>cntr, rel</i>	$(cntr) - 1 \Rightarrow cntr$ If $(cntr) \neq 0$ , then Branch; else Continue to next instruction  Decrement Counter and Branch if $\neq 0$ ( $cntr = A, B, D, X, Y$ , or SP)	REL (9-bit)	04 lb rr	3	-	-	-	-	-	-	-	-

Table A-1 Instruction Set Summary (Continued)

Source Form	Operation	Addr. Mode	Machine Coding (hex)	~*	S	X	H	I	N	Z	V	C
DEC <i>opr</i>	(M) – \$01 ⇒ M Decrement Memory Location	EXT	73 hh ll	4	–	–	–	–	Δ	Δ	Δ	–
		IDX	63 xb	3								
		IDX1	63 xb ff	4								
		IDX2	63 xb ee ff	5								
		[D,IDX]	63 xb	6								
		[IDX2]	63 xb ee ff	6								
DECA	(A) – \$01 ⇒ A      Decrement A	INH	43	1								
DECB	(B) – \$01 ⇒ B      Decrement B	INH	53	1								
DES	(SP) – \$0001 ⇒ SP <i>Translates to LEAS –1,SP</i>	IDX	1B 9F	2	–	–	–	–	–	–	–	–
DEX	(X) – \$0001 ⇒ X Decrement Index Register X	INH	09	1	–	–	–	–	–	Δ	–	–
DEY	(Y) – \$0001 ⇒ Y Decrement Index Register Y	INH	03	1	–	–	–	–	–	Δ	–	–
EDIV	(Y:D) ÷ (X) ⇒ Y Remainder ⇒ D 32 × 16 Bit ⇒ 16 Bit Divide (unsigned)	INH	11	11	–	–	–	–	Δ	Δ	Δ	Δ
EDIVS	(Y:D) ÷ (X) ⇒ Y Remainder ⇒ D 32 × 16 Bit ⇒ 16 Bit Divide (signed)	INH	18 14	12	–	–	–	–	Δ	Δ	Δ	Δ
EMACS <i>sum</i>	(M <sub>(X)</sub> :M <sub>(X+1)</sub> ) × (M <sub>(Y)</sub> :M <sub>(Y+1)</sub> ) + (M–M+3) ⇒ M–M+3  16 × 16 Bit ⇒ 32 Bit Multiply and Accumulate (signed)	Special	18 12 hh ll	13	–	–	–	–	Δ	Δ	Δ	Δ
EMAXD <i>opr</i>	MAX((D), (M:M+1)) ⇒ D MAX of 2 Unsigned 16-Bit Values  N, Z, V and C status bits reflect result of internal compare ((D) – (M:M+1))	IDX	18 1A xb	4	–	–	–	–	Δ	Δ	Δ	Δ
		IDX1	18 1A xb ff	4								
		IDX2	18 1A xb ee ff	5								
		[D,IDX]	18 1A xb	7								
		[IDX2]	18 1A xb ee ff	7								
EMAXM <i>opr</i>	MAX((D), (M:M+1)) ⇒ M:M+1 MAX of 2 Unsigned 16-Bit Values  N, Z, V and C status bits reflect result of internal compare ((D) – (M:M+1))	IDX	18 1E xb	4	–	–	–	–	Δ	Δ	Δ	Δ
		IDX1	18 1E xb ff	5								
		IDX2	18 1E xb ee ff	6								
		[D,IDX]	18 1E xb	7								
		[IDX2]	18 1E xb ee ff	7								
EMIND <i>opr</i>	MIN((D), (M:M+1)) ⇒ D MIN of 2 Unsigned 16-Bit Values  N, Z, V and C status bits reflect result of internal compare ((D) – (M:M+1))	IDX	18 1B xb	4	–	–	–	–	Δ	Δ	Δ	Δ
		IDX1	18 1B xb ff	4								
		IDX2	18 1B xb ee ff	5								
		[D,IDX]	18 1B xb	7								
		[IDX2]	18 1B xb ee ff	7								
EMINM <i>opr</i>	MIN((D), (M:M+1)) ⇒ M:M+1 MIN of 2 Unsigned 16-Bit Values  N, Z, V and C status bits reflect result of internal compare ((D) – (M:M+1))	IDX	18 1F xb	4	–	–	–	–	Δ	Δ	Δ	Δ
		IDX1	18 1F xb ff	5								
		IDX2	18 1F xb ee ff	6								
		[D,IDX]	18 1F xb	7								
		[IDX2]	18 1F xb ee ff	7								
EMUL	(D) × (Y) ⇒ Y:D 16 × 16 Bit Multiply (unsigned)	INH	13	3	–	–	–	–	Δ	Δ	–	Δ
EMULS	(D) × (Y) ⇒ Y:D 16 × 16 Bit Multiply (signed)	INH	18 13	3	–	–	–	–	Δ	Δ	–	Δ

**Table A-1 Instruction Set Summary (Continued)**

Source Form	Operation	Addr. Mode	Machine Coding (hex)	~*	S	X	H	I	N	Z	V	C
EORA <i>opr</i>	(A) $\oplus$ (M) $\Rightarrow$ A Exclusive-OR A with Memory	IMM	88 ii	1	—	—	—	—	$\Delta$	$\Delta$	0	—
		DIR	98 dd	3	—	—	—	—				
		EXT	B8 hh ll	3	—	—	—	—				
		IDX	A8 xb	3	—	—	—	—				
		IDX1	A8 xb ff	3	—	—	—	—				
		IDX2	A8 xb ee ff	4	—	—	—	—				
		[D,IDX] [IDX2]	A8 xb A8 xb ee ff	6 6	—	—	—	—				
EORB <i>opr</i>	(B) $\oplus$ (M) $\Rightarrow$ B Exclusive-OR B with Memory	IMM	C8 ii	1	—	—	—	—	$\Delta$	$\Delta$	0	—
		DIR	D8 dd	3	—	—	—	—				
		EXT	F8 hh ll	3	—	—	—	—				
		IDX	E8 xb	3	—	—	—	—				
		IDX1	E8 xb ff	3	—	—	—	—				
		IDX2	E8 xb ee ff	4	—	—	—	—				
		[D,IDX] [IDX2]	E8 xb E8 xb ee ff	6 6	—	—	—	—				
ETBL <i>opr</i>	(M:M+1) + [(B) $\times$ ((M+2:M+3) - (M:M+1))] $\Rightarrow$ D 16-Bit Table Lookup and Interpolate  Initialize B, and index before ETBL. <ea> points at first table entry (M:M+1) and B is fractional part of lookup value  (no indirect addr. modes allowed)	IDX	18 3F xb	10	—	—	—	—	$\Delta$	$\Delta$	—	?
EXG <i>r1, r2</i>	(r1) $\Leftrightarrow$ (r2) (if r1 and r2 same size) or \$00:(r1) $\Rightarrow$ r2 (if r1=8-bit; r2=16-bit) or (r1 <sub>low</sub> ) $\Leftrightarrow$ (r2) (if r1=16-bit; r2=8-bit)  r1 and r2 may be A, B, CCR, D, X, Y, or SP	INH	B7 eb	1	—	—	—	—	—	—	—	—
FDIV	(D) $\div$ (X) $\Rightarrow$ X; r $\Rightarrow$ D 16 $\times$ 16 Bit Fractional Divide	INH	18 11	12	—	—	—	—	—	$\Delta$	$\Delta$	$\Delta$
IBEQ <i>cntr, rel</i>	(cntr) + 1 $\Rightarrow$ cntr If (cntr) = 0, then Branch else Continue to next instruction  Increment Counter and Branch if = 0 (cntr = A, B, D, X, Y, or SP)	REL (9-bit)	04 lb rr	3	—	—	—	—	—	—	—	—
IBNE <i>cntr, rel</i>	(cntr) + 1 $\Rightarrow$ cntr if (cntr) not = 0, then Branch; else Continue to next instruction  Increment Counter and Branch if $\neq$ 0 (cntr = A, B, D, X, Y, or SP)	REL (9-bit)	04 lb rr	3	—	—	—	—	—	—	—	—
IDIV	(D) $\div$ (X) $\Rightarrow$ X; r $\Rightarrow$ D 16 $\times$ 16 Bit Integer Divide (unsigned)	INH	18 10	12	—	—	—	—	—	$\Delta$	0	$\Delta$
IDIVS	(D) $\div$ (X) $\Rightarrow$ X; r $\Rightarrow$ D 16 $\times$ 16 Bit Integer Divide (signed)	INH	18 15	12	—	—	—	—	$\Delta$	$\Delta$	$\Delta$	$\Delta$

Table A-1 Instruction Set Summary (Continued)

Source Form	Operation	Addr. Mode	Machine Coding (hex)	~*	S	X	H	I	N	Z	V	C
INC <i>opr</i>	(M) + \$01 ⇒ M Increment Memory Byte	EXT	72 hh ll	4	—	—	—	—	Δ	Δ	Δ	—
		IDX	62 xb	3	—	—	—	—	—	—	—	—
		IDX1	62 xb ff	4	—	—	—	—	—	—	—	—
		IDX2	62 xb ee ff	5	—	—	—	—	—	—	—	—
		[D,IDX]	62 xb	6	—	—	—	—	—	—	—	—
		[IDX2]	62 xb ee ff	6	—	—	—	—	—	—	—	—
INCA	(A) + \$01 ⇒ A      Increment Acc. A	INH	42	1	—	—	—	—	—	—	—	—
INCB	(B) + \$01 ⇒ B      Increment Acc. B	INH	52	1	—	—	—	—	—	—	—	—
INS	(SP) + \$0001 ⇒ SP <i>Translates to LEAS 1,SP</i>	IDX	1B 81	2	—	—	—	—	—	—	—	—
INX	(X) + \$0001 ⇒ X Increment Index Register X	INH	08	1	—	—	—	—	—	Δ	—	—
INY	(Y) + \$0001 ⇒ Y Increment Index Register Y	INH	02	1	—	—	—	—	—	Δ	—	—
JMP <i>opr</i>	Subroutine address ⇒ PC  Jump	EXT	06 hh ll	3	—	—	—	—	—	—	—	—
		IDX	05 xb	3	—	—	—	—	—	—	—	—
		IDX1	05 xb ff	3	—	—	—	—	—	—	—	—
		IDX2	05 xb ee ff	4	—	—	—	—	—	—	—	—
		[D,IDX]	05 xb	6	—	—	—	—	—	—	—	—
		[IDX2]	05 xb ee ff	6	—	—	—	—	—	—	—	—
JSR <i>opr</i>	(SP) – 2 ⇒ SP; RTN <sub>H</sub> :RTN <sub>L</sub> ⇒ M <sub>(SP):M<sub>(SP+1)</sub></sub> ; Subroutine address ⇒ PC  Jump to Subroutine	DIR	17 dd	4	—	—	—	—	—	—	—	—
		EXT	16 hh ll	4	—	—	—	—	—	—	—	—
		IDX	15 xb	4	—	—	—	—	—	—	—	—
		IDX1	15 xb ff	4	—	—	—	—	—	—	—	—
		IDX2	15 xb ee ff	5	—	—	—	—	—	—	—	—
		[D,IDX]	15 xb	7	—	—	—	—	—	—	—	—
		[IDX2]	15 xb ee ff	7	—	—	—	—	—	—	—	—
LBCC <i>rel</i>	Long Branch if Carry Clear (if C = 0)	REL	18 24 qq rr	4/3	—	—	—	—	—	—	—	—
LBCS <i>rel</i>	Long Branch if Carry Set (if C = 1)	REL	18 25 qq rr	4/3	—	—	—	—	—	—	—	—
LBEQ <i>rel</i>	Long Branch if Equal (if Z = 1)	REL	18 27 qq rr	4/3	—	—	—	—	—	—	—	—
LBGE <i>rel</i>	Long Branch Greater Than or Equal (if N ⊕ V = 0) (signed)	REL	18 2C qq rr	4/3	—	—	—	—	—	—	—	—
LBGT <i>rel</i>	Long Branch if Greater Than (if Z + (N ⊕ V) = 0) (signed)	REL	18 2E qq rr	4/3	—	—	—	—	—	—	—	—
LBHI <i>rel</i>	Long Branch if Higher (if C + Z = 0) (unsigned)	REL	18 22 qq rr	4/3	—	—	—	—	—	—	—	—
LBHS <i>rel</i>	Long Branch if Higher or Same (if C = 0) (unsigned) same function as LBCC	REL	18 24 qq rr	4/3	—	—	—	—	—	—	—	—
LBLE <i>rel</i>	Long Branch if Less Than or Equal (if Z + (N ⊕ V) = 1) (signed)	REL	18 2F qq rr	4/3	—	—	—	—	—	—	—	—
LBLO <i>rel</i>	Long Branch if Lower (if C = 1) (unsigned) same function as LBCS	REL	18 25 qq rr	4/3	—	—	—	—	—	—	—	—
LBLS <i>rel</i>	Long Branch if Lower or Same (if C + Z = 1) (unsigned)	REL	18 23 qq rr	4/3	—	—	—	—	—	—	—	—
LBLT <i>rel</i>	Long Branch if Less Than (if N ⊕ V = 1) (signed)	REL	18 2D qq rr	4/3	—	—	—	—	—	—	—	—
LBMI <i>rel</i>	Long Branch if Minus (if N = 1)	REL	18 2B qq rr	4/3	—	—	—	—	—	—	—	—
LBNE <i>rel</i>	Long Branch if Not Equal (if Z = 0)	REL	18 26 qq rr	4/3	—	—	—	—	—	—	—	—
LBPL <i>rel</i>	Long Branch if Plus (if N = 0)	REL	18 2A qq rr	4/3	—	—	—	—	—	—	—	—
LBRA <i>rel</i>	Long Branch Always (if 1=1)	REL	18 20 qq rr	4	—	—	—	—	—	—	—	—

Table A-1 Instruction Set Summary (Continued)

Source Form	Operation	Addr. Mode	Machine Coding (hex)	~*	S	X	H	I	N	Z	V	C
LBRN <i>rel</i>	Long Branch Never (if 1 = 0)	REL	18 21 qq rr	3	-	-	-	-	-	-	-	-
LBVC <i>rel</i>	Long Branch if Overflow Bit Clear (if V=0)	REL	18 28 qq rr	4/3	-	-	-	-	-	-	-	-
LBVS <i>rel</i>	Long Branch if Overflow Bit Set (if V = 1)	REL	18 29 qq rr	4/3	-	-	-	-	-	-	-	-
LDAA <i>opr</i>	(M) ⇒ A Load Accumulator A	IMM	86 ii	1	-	-	-	-	Δ	Δ	0	-
		DIR	96 dd	3								
		EXT	B6 hh ll	3								
		IDX	A6 xb	3								
		IDX1	A6 xb ff	3								
		IDX2	A6 xb ee ff	4								
		[D,IDX] [IDX2]	A6 xb A6 xb ee ff	6 6								
LDAB <i>opr</i>	(M) ⇒ B Load Accumulator B	IMM	C6 ii	1	-	-	-	-	Δ	Δ	0	-
		DIR	D6 dd	3								
		EXT	F6 hh ll	3								
		IDX	E6 xb	3								
		IDX1	E6 xb ff	3								
		IDX2	E6 xb ee ff	4								
		[D,IDX] [IDX2]	E6 xb E6 xb ee ff	6 6								
LDD <i>opr</i>	(M:M+1) ⇒ A:B Load Double Accumulator D (A:B)	IMM	CC jj kk	2	-	-	-	-	Δ	Δ	0	-
		DIR	DC dd	3								
		EXT	FC hh ll	3								
		IDX	EC xb	3								
		IDX1	EC xb ff	3								
		IDX2	EC xb ee ff	4								
		[D,IDX] [IDX2]	EC xb EC xb ee ff	6 6								
LDS <i>opr</i>	(M:M+1) ⇒ SP Load Stack Pointer	IMM	CF jj kk	2	-	-	-	-	Δ	Δ	0	-
		DIR	DF dd	3								
		EXT	FF hh ll	3								
		IDX	EF xb	3								
		IDX1	EF xb ff	3								
		IDX2	EF xb ee ff	4								
		[D,IDX] [IDX2]	EF xb EF xb ee ff	6 6								
LDX <i>opr</i>	(M:M+1) ⇒ X Load Index Register X	IMM	CE jj kk	2	-	-	-	-	Δ	Δ	0	-
		DIR	DE dd	3								
		EXT	FE hh ll	3								
		IDX	EE xb	3								
		IDX1	EE xb ff	3								
		IDX2	EE xb ee ff	4								
		[D,IDX] [IDX2]	EE xb EE xb ee ff	6 6								
LDY <i>opr</i>	(M:M+1) ⇒ Y Load Index Register Y	IMM	CD jj kk	2	-	-	-	-	Δ	Δ	0	-
		DIR	DD dd	3								
		EXT	FD hh ll	3								
		IDX	ED xb	3								
		IDX1	ED xb ff	3								
		IDX2	ED xb ee ff	4								
		[D,IDX] [IDX2]	ED xb ED xb ee ff	6 6								
LEAS <i>opr</i>	Effective Address ⇒ SP Load Effective Address into SP	IDX	1B xb	2	-	-	-	-	-	-	-	-
		IDX1	1B xb ff	2								
		IDX2	1B xb ee ff	2								



Table A-1 Instruction Set Summary (Continued)

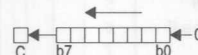
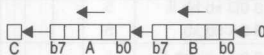
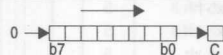
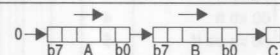
Source Form	Operation	Addr. Mode	Machine Coding (hex)	~*	S	X	H	I	N	Z	V	C
LEAX <i>opr</i>	Effective Address $\Rightarrow$ X Load Effective Address into X	IDX IDX1 IDX2	1A xb 1A xb ff 1A xb ee ff	2 2 2	-	-	-	-	-	-	-	-
LEAY <i>opr</i>	Effective Address $\Rightarrow$ Y Load Effective Address into Y	IDX IDX1 IDX2	19 xb 19 xb ff 19 xb ee ff	2 2 2	-	-	-	-	-	-	-	-
LSL <i>opr</i>	 Logical Shift Left same function as ASL	EXT IDX IDX1 IDX2 [D,IDX] [IDX2]	78 hh ll 68 xb 68 xb ff 68 xb ee ff 68 xb 68 xb ee ff	4 3 4 5 6 6	-	-	-	-	$\Delta$	$\Delta$	$\Delta$	
LSLA LSLB	Logical Shift Accumulator A to Left Logical Shift Accumulator B to Left	INH INH	48 58	1 1								
LSLD	 Logical Shift Left D Accumulator same function as ASDL	INH	59	1	-	-	-	-	$\Delta$	$\Delta$	$\Delta$	$\Delta$
LSR <i>opr</i>	 Logical Shift Right	EXT IDX IDX1 IDX2 [D,IDX] [IDX2]	74 hh ll 64 xb 64 xb ff 64 xb ee ff 64 xb 64 xb ee ff	4 3 4 5 6 6	-	-	-	-	0	$\Delta$	$\Delta$	$\Delta$
LSRA LSRB	Logical Shift Accumulator A to Right Logical Shift Accumulator B to Right	INH INH	44 54	1 1								
LSRD	 Logical Shift Right D Accumulator	INH	49	1	-	-	-	-	0	$\Delta$	$\Delta$	$\Delta$
MAXA	MAX((A), (M)) $\Rightarrow$ A MAX of 2 Unsigned 8-Bit Values  N, Z, V and C status bits reflect result of internal compare ((A) - (M)).	IDX IDX1 IDX2 [D,IDX] [IDX2]	18 18 xb 18 18 xb ff 18 18 xb ee ff 18 18 xb 18 18 xb ee ff	4 4 5 7 7	-	-	-	-	$\Delta$	$\Delta$	$\Delta$	$\Delta$
MAXM	MAX((A), (M)) $\Rightarrow$ M MAX of 2 Unsigned 8-Bit Values  N, Z, V and C status bits reflect result of internal compare ((A) - (M)).	IDX IDX1 IDX2 [D,IDX] [IDX2]	18 1C xb 18 1C xb ff 18 1C xb ee ff 18 1C xb 18 1C xb ee ff	4 5 6 7 7	-	-	-	-	$\Delta$	$\Delta$	$\Delta$	$\Delta$
MEM	$\mu$ (grade) $\Rightarrow$ M(Y); (X) + 4 $\Rightarrow$ X; (Y) + 1 $\Rightarrow$ Y; A unchanged  if (A) < P1 or (A) > P2 then $\mu = 0$ , else $\mu = \text{MIN}[(\text{(A) - P1}) \times \text{S1}, (\text{P2 - (A)}) \times \text{S2}, \$\text{FF}]$ where: A = current crisp input value; X points at 4-byte data structure that describes a trapezoidal membership function (P1, P2, S1, S2); Y points at fuzzy input (RAM location). See instruction details for special cases.	Special	01	5	-	-	?	-	?	?	?	?



Table A-1 Instruction Set Summary (Continued)

Source Form	Operation	Addr. Mode	Machine Coding (hex)	~*	S	X	H	I	N	Z	V	C
MINA	MIN((A), (M)) $\Rightarrow$ A MIN of Two Unsigned 8-Bit Values	IDX	18 19 xb	4	-	-	-	-	$\Delta$	$\Delta$	$\Delta$	$\Delta$
		IDX1	18 19 xb ff	4								
		IDX2	18 19 xb ee ff	5								
	N, Z, V and C status bits reflect result of internal compare ((A) - (M)).	[D,IDX]	18 19 xb	7								
		[IDX2]	18 19 xb ee ff	7								
MINM	MIN((A), (M)) $\Rightarrow$ M MIN of Two Unsigned 8-Bit Values	IDX	18 1D xb	4	-	-	-	-	$\Delta$	$\Delta$	$\Delta$	$\Delta$
		IDX1	18 1D xb ff	5								
		IDX2	18 1D xb ee ff	6								
	N, Z, V and C status bits reflect result of internal compare ((A) - (M)).	[D,IDX]	18 1D xb	7								
		[IDX2]	18 1D xb ee ff	7								
MOVB <i>opr1, opr2</i>	(M <sub>1</sub> ) $\Rightarrow$ M <sub>2</sub> Memory to Memory Byte-Move (8-Bit)	IMM-EXT	18 0B ii hh ll	4	-	-	-	-	-	-	-	-
		IMM-IDX	18 08 xb ii	4								
		EXT-EXT	18 0C hh ll hh ll	6								
		EXT-IDX	18 09 xb hh ll	5								
		IDX-EXT	18 0D xb hh ll	5								
		IDX-IDX	18 0A xb xb	5								
MOVW <i>opr1, opr2</i>	(M:M+1 <sub>1</sub> ) $\Rightarrow$ M:M+1 <sub>2</sub> Memory to Memory Word-Move (16-Bit)	IMM-EXT	18 03 jj kk hh ll	5	-	-	-	-	-	-	-	-
		IMM-IDX	18 00 xb jj kk	4								
		EXT-EXT	18 04 hh ll hh ll	6								
		EXT-IDX	18 01 xb hh ll	5								
		IDX-EXT	18 05 xb hh ll	5								
		IDX-IDX	18 02 xb xb	5								
MUL	(A) $\times$ (B) $\Rightarrow$ A:B	INH	12	3	-	-	-	-	-	-	-	$\Delta$
	8 $\times$ 8 Unsigned Multiply											
NEG <i>opr</i>	0 - (M) $\Rightarrow$ M or (M) + 1 $\Rightarrow$ M Two's Complement Negate	EXT	70 hh ll	4	-	-	-	-	$\Delta$	$\Delta$	$\Delta$	$\Delta$
		IDX	60 xb	3								
		IDX1	60 xb ff	4								
		IDX2	60 xb ee ff	5								
		[D,IDX]	60 xb	6								
NEGA	0 - (A) $\Rightarrow$ A equivalent to (A) + 1 $\Rightarrow$ B Negate Accumulator A	[IDX2]	60 xb ee ff	6								
		INH	40	1								
NEGB	0 - (B) $\Rightarrow$ B equivalent to (B) + 1 $\Rightarrow$ B Negate Accumulator B	INH	50	1								
NOP	No Operation	INH	A7	1	-	-	-	-	-	-	-	-
ORAA <i>opr</i>	(A) + (M) $\Rightarrow$ A Logical OR A with Memory	IMM	8A ii	1	-	-	-	-	$\Delta$	$\Delta$	0	-
		DIR	9A dd	3								
		EXT	BA hh ll	3								
		IDX	AA xb	3								
		IDX1	AA xb ff	3								
		IDX2	AA xb ee ff	4								
		[D,IDX]	AA xb	6								
		[IDX2]	AA xb ee ff	6								
ORAB <i>opr</i>	(B) + (M) $\Rightarrow$ B Logical OR B with Memory	IMM	CA ii	1	-	-	-	-	$\Delta$	$\Delta$	0	-
		DIR	DA dd	3								
		EXT	FA hh ll	3								
		IDX	EA xb	3								
		IDX1	EA xb ff	3								
		IDX2	EA xb ee ff	4								
		[D,IDX]	EA xb	6								
		[IDX2]	EA xb ee ff	6								
ORCC <i>opr</i>	(CCR) + M $\Rightarrow$ CCR Logical OR CCR with Memory	IMM	14 ii	1	$\uparrow$	-	$\uparrow$	$\uparrow$	$\uparrow$	$\uparrow$	$\uparrow$	$\uparrow$

Table A-1 Instruction Set Summary (Continued)

Source Form	Operation	Addr. Mode	Machine Coding (hex)	~*	S	X	H	I	N	Z	V	C
PSHA	(SP) - 1 $\Rightarrow$ SP; (A) $\Rightarrow$ M <sub>(SP)</sub> Push Accumulator A onto Stack	INH	36	2	-	-	-	-	-	-	-	-
PSHB	(SP) - 1 $\Rightarrow$ SP; (B) $\Rightarrow$ M <sub>(SP)</sub> Push Accumulator B onto Stack	INH	37	2	-	-	-	-	-	-	-	-
PSHC	(SP) - 1 $\Rightarrow$ SP; (CCR) $\Rightarrow$ M <sub>(SP)</sub> Push CCR onto Stack	INH	39	2	-	-	-	-	-	-	-	-
PSHD	(SP) - 2 $\Rightarrow$ SP; (A:B) $\Rightarrow$ M <sub>(SP)</sub> :M <sub>(SP+1)</sub> Push D Accumulator onto Stack	INH	3B	2	-	-	-	-	-	-	-	-
PSHX	(SP) - 2 $\Rightarrow$ SP; (X <sub>H</sub> :X <sub>L</sub> ) $\Rightarrow$ M <sub>(SP)</sub> :M <sub>(SP+1)</sub> Push Index Register X onto Stack	INH	34	2	-	-	-	-	-	-	-	-
PSHY	(SP) - 2 $\Rightarrow$ SP; (Y <sub>H</sub> :Y <sub>L</sub> ) $\Rightarrow$ M <sub>(SP)</sub> :M <sub>(SP+1)</sub> Push Index Register Y onto Stack	INH	35	2	-	-	-	-	-	-	-	-
PULA	(M <sub>(SP)</sub> ) $\Rightarrow$ A; (SP) + 1 $\Rightarrow$ SP Pull Accumulator A from Stack	INH	32	3	-	-	-	-	-	-	-	-
PULB	(M <sub>(SP)</sub> ) $\Rightarrow$ B; (SP) + 1 $\Rightarrow$ SP Pull Accumulator B from Stack	INH	33	3	-	-	-	-	-	-	-	-
PULC	(M <sub>(SP)</sub> ) $\Rightarrow$ CCR; (SP) + 1 $\Rightarrow$ SP Pull CCR from Stack	INH	38	3	$\Delta$	$\Downarrow$	$\Delta$	$\Delta$	$\Delta$	$\Delta$	$\Delta$	$\Delta$
PULD	(M <sub>(SP)</sub> :M <sub>(SP+1)</sub> ) $\Rightarrow$ A:B; (SP) + 2 $\Rightarrow$ SP Pull D from Stack	INH	3A	3	-	-	-	-	-	-	-	-
PULX	(M <sub>(SP)</sub> :M <sub>(SP+1)</sub> ) $\Rightarrow$ X <sub>H</sub> :X <sub>L</sub> ; (SP) + 2 $\Rightarrow$ SP Pull Index Register X from Stack	INH	30	3	-	-	-	-	-	-	-	-
PULY	(M <sub>(SP)</sub> :M <sub>(SP+1)</sub> ) $\Rightarrow$ Y <sub>H</sub> :Y <sub>L</sub> ; (SP) + 2 $\Rightarrow$ SP Pull Index Register Y from Stack	INH	31	3	-	-	-	-	-	-	-	-
REV	MIN-MAX rule evaluation Find smallest rule input (MIN). Store to rule outputs unless fuzzy output is already larger (MAX).  For rule weights see REVW.  Each rule input is an 8-bit offset from the base address in Y. Each rule output is an 8-bit offset from the base address in Y. \$FE separates rule inputs from rule outputs. \$FF terminates the rule list.  REV may be interrupted.	Special	18 3A	3** per rule byte	-	-	-	-	-	-	$\Delta$	-

Table A-1 Instruction Set Summary (Continued)

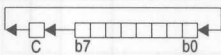
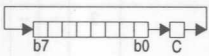
Source Form	Operation	Addr. Mode	Machine Coding (hex)	~*	S	X	H	I	N	Z	V	C
REVV	MIN-MAX rule evaluation Find smallest rule input (MIN), Store to rule outputs unless fuzzy output is already larger (MAX).  Rule weights supported, optional.  Each rule input is the 16-bit address of a fuzzy input. Each rule output is the 16-bit ad- dress of a fuzzy output. The value \$FFE separates rule inputs from rule outputs. \$FFF terminates the rule list.  REVV may be interrupted.	Special	18 3B	3 <sup>''</sup> per rule byte; 5 per wt.	-	-	?	-	?	?	Δ	!
ROL <i>opr</i>	 Rotate Memory Left through Carry	EXT IDX IDX1 IDX2 [D,IDX] [IDX2] INH INH	75 hh ll 65 xb 65 xb ff 65 xb ee ff 65 xb 65 xb ee ff 45 55	4 3 4 5 6 6 1 1	-	-	-	-	Δ	Δ	Δ	
ROLA ROLB	Rotate A Left through Carry Rotate B Left through Carry											
ROR <i>opr</i>	 Rotate Memory Right through Carry	EXT IDX IDX1 IDX2 [D,IDX] [IDX2] INH INH	76 hh ll 66 xb 66 xb ff 66 xb ee ff 66 xb 66 xb ee ff 46 56	4 3 4 5 6 6 1 1	-	-	-	-	Δ	Δ	Δ	
RORA RORB	Rotate A Right through Carry Rotate B Right through Carry											
RTC	(M <sub>(SP)</sub> ) ⇒ PPAGE; (SP) + 1 ⇒ SP; (M <sub>(SP)</sub> :M <sub>(SP+1)</sub> ) ⇒ PC <sub>H</sub> :PC <sub>L</sub> ; (SP) + 2 ⇒ SP  Return from Call	INH	0A	6	-	-	-	-	-	-	-	-
RTI	(M <sub>(SP)</sub> ) ⇒ CCR; (SP) + 1 ⇒ SP (M <sub>(SP)</sub> :M <sub>(SP+1)</sub> ) ⇒ B:A; (SP) + 2 ⇒ SP (M <sub>(SP)</sub> :M <sub>(SP+1)</sub> ) ⇒ X <sub>H</sub> :X <sub>L</sub> ; (SP) + 4 ⇒ SP (M <sub>(SP)</sub> :M <sub>(SP+1)</sub> ) ⇒ PC <sub>H</sub> :PC <sub>L</sub> ; (SP) - 2 ⇒ SP (M <sub>(SP)</sub> :M <sub>(SP+1)</sub> ) ⇒ Y <sub>H</sub> :Y <sub>L</sub> ; (SP) + 4 ⇒ SP  Return from Interrupt	INH	0B	8	Δ	↓	Δ	Δ	Δ	Δ	Δ	Δ
RTS	(M <sub>(SP)</sub> :M <sub>(SP+1)</sub> ) ⇒ PC <sub>H</sub> :PC <sub>L</sub> ; (SP) + 2 ⇒ SP  Return from Subroutine	INH	3D	5	-	-	-	-	-	-	-	-
SBA	(A) - (B) ⇒ A Subtract B from A	INH	18 16	2	-	-	-	-	Δ	Δ	Δ	Δ

Table A-1 Instruction Set Summary (Continued)

Source Form	Operation	Addr. Mode	Machine Coding (hex)	~*	S	X	H	I	N	Z	V	C
SBCA <i>opr</i>	(A) – (M) – C ⇒ A Subtract with Borrow from A	IMM	82 ii	1	–	–	–	–	Δ	Δ	Δ	Δ
		DIR	92 dd	3								
		EXT	B2 hh ll	3								
		IDX	A2 xb	3								
		IDX1	A2 xb ff	3								
		IDX2	A2 xb ee ff	4								
		[D,IDX] [IDX2]	A2 xb A2 xb ee ff	6 6								
SBCB <i>opr</i>	(B) – (M) – C ⇒ B Subtract with Borrow from B	IMM	C2 ii	1	–	–	–	–	Δ	Δ	Δ	Δ
		DIR	D2 dd	3								
		EXT	F2 hh ll	3								
		IDX	E2 xb	3								
		IDX1	E2 xb ff	3								
		IDX2	E2 xb ee ff	4								
		[D,IDX] [IDX2]	E2 xb E2 xb ee ff	6 6								
SEC	1 ⇒ C <i>Translates to ORCC #01</i>	IMM	14 01	1	–	–	–	–	–	–	–	1
SEI	1 ⇒ I; (inhibit I interrupts) <i>Translates to ORCC #10</i>	IMM	14 10	1	–	–	–	1	–	–	–	–
SEV	1 ⇒ V <i>Translates to ORCC #02</i>	IMM	14 02	1	–	–	–	–	–	–	1	–
SEX <i>r1, r2</i>	\$00:(r1) ⇒ r2 if r1, bit 7 is 0 <i>or</i> \$FF:(r1) ⇒ r2 if r1, bit 7 is 1  Sign Extend 8-bit r1 to 16-bit r2 r1 may be A, B, or CCR r2 may be D, X, Y, or SP  <i>Alternate mnemonic for TFR r1, r2</i>	INH	B7 eb	1	–	–	–	–	–	–	–	–
STAA <i>opr</i>	(A) ⇒ M Store Accumulator A to Memory	DIR	5A dd	2	–	–	–	–	Δ	Δ	0	–
		EXT	7A hh ll	3								
		IDX	6A xb	2								
		IDX1	6A xb ff	3								
		IDX2	6A xb ee ff	3								
		[D,IDX]	6A xb	5								
		[IDX2]	6A xb ee ff	5								
STAB <i>opr</i>	(B) ⇒ M Store Accumulator B to Memory	DIR	5B dd	2	–	–	–	–	Δ	Δ	0	–
		EXT	7B hh ll	3								
		IDX	6B xb	2								
		IDX1	6B xb ff	3								
		IDX2	6B xb ee ff	3								
		[D,IDX]	6B xb	5								
		[IDX2]	6B xb ee ff	5								
STD <i>opr</i>	(A) ⇒ M, (B) ⇒ M+1 Store Double Accumulator	DIR	5C dd	2	–	–	–	–	Δ	Δ	0	–
		EXT	7C hh ll	3								
		IDX	6C xb	2								
		IDX1	6C xb ff	3								
		IDX2	6C xb ee ff	3								
		[D,IDX]	6C xb	5								
		[IDX2]	6C xb ee ff	5								

Table A-1 Instruction Set Summary (Continued)

Source Form	Operation	Addr. Mode	Machine Coding (hex)	~*	S	X	H	I	N	Z	V	C
STOP	$(SP) - 2 \Rightarrow SP;$ $RTN_H:RTN_L \Rightarrow M_{(SP)}:M_{(SP+1)};$ $(SP) - 2 \Rightarrow SP; (Y_H:Y_L) \Rightarrow M_{(SP)}:M_{(SP+1)};$ $(SP) - 2 \Rightarrow SP; (X_H:X_L) \Rightarrow M_{(SP)}:M_{(SP+1)};$ $(SP) - 2 \Rightarrow SP; (B:A) \Rightarrow M_{(SP)}:M_{(SP+1)};$ $(SP) - 1 \Rightarrow SP; (CCR) \Rightarrow M_{(SP)};$ STOP All Clocks  If S control bit = 1, the STOP instruction is disabled and acts like a two-cycle NOP.  Registers stacked to allow quicker recovery by interrupt.	INH	18 3E	9** +5 or +2**	-	-	-	-	-	-	-	-
STS <i>opr</i>	$(SP_H:SP_L) \Rightarrow M:M+1$ Store Stack Pointer	DIR EXT IDX IDX1 IDX2 [D,IDX] [IDX2]	5F dd 7F hh ll 6F xb 6F xb ff 6F xb ee ff 6F xb 6F xb ee ff	2 3 2 3 3 5 5	-	-	-	-	$\Delta$	$\Delta$	0	-
STX <i>opr</i>	$(X_H:X_L) \Rightarrow M:M+1$ Store Index Register X	DIR EXT IDX IDX1 IDX2 [D,IDX] [IDX2]	5E dd 7E hh ll 6E xb 6E xb ff 6E xb ee ff 6E xb 6E xb ee ff	2 3 2 3 3 5 5	-	-	-	-	$\Delta$	$\Delta$	0	-
STY <i>opr</i>	$(Y_H:Y_L) \Rightarrow M:M+1$ Store Index Register Y	DIR EXT IDX IDX1 IDX2 [D,IDX] [IDX2]	5D dd 7D hh ll 6D xb 6D xb ff 6D xb ee ff 6D xb 6D xb ee ff	2 3 2 3 3 5 5	-	-	-	-	$\Delta$	$\Delta$	0	-
SUBA <i>opr</i>	$(A) - (M) \Rightarrow A$ Subtract Memory from Accumulator A	IMM DIR EXT IDX IDX1 IDX2 [D,IDX] [IDX2]	80 ii 90 dd B0 hh ll A0 xb A0 xb ff A0 xb ee ff A0 xb A0 xb ee ff	1 3 3 3 3 4 6 6	-	-	-	-	$\Delta$	$\Delta$	$\Delta$	$\Delta$
SUBB <i>opr</i>	$(B) - (M) \Rightarrow B$ Subtract Memory from Accumulator B	IMM DIR EXT IDX IDX1 IDX2 [D,IDX] [IDX2]	C0 ii D0 dd F0 hh ll E0 xb E0 xb ff E0 xb ee ff E0 xb E0 xb ee ff	1 3 3 3 3 4 6 6	-	-	-	-	$\Delta$	$\Delta$	$\Delta$	$\Delta$

Table A-1 Instruction Set Summary (Continued)

Source Form	Operation	Addr. Mode	Machine Coding (hex)	~*	S	X	H	I	N	Z	V	C
SUBD <i>opr</i>	(D) - (M:M+1) $\Rightarrow$ D Subtract Memory from D (A:B)	IMM	83 jj kk	2	-	-	-	-	$\Delta$	$\Delta$	$\Delta$	$\Delta$
		DIR	93 dd	3	-	-	-	-	$\Delta$	$\Delta$	$\Delta$	$\Delta$
		EXT	B3 hh ll	3	-	-	-	-	$\Delta$	$\Delta$	$\Delta$	$\Delta$
		IDX	A3 xb	3	-	-	-	-	$\Delta$	$\Delta$	$\Delta$	$\Delta$
		IDX1	A3 xb ff	3	-	-	-	-	$\Delta$	$\Delta$	$\Delta$	$\Delta$
		IDX2	A3 xb ee ff	4	-	-	-	-	$\Delta$	$\Delta$	$\Delta$	$\Delta$
		[D,IDX] [IDX2]	A3 xb A3 xb ee ff	6 6	-	-	-	-	$\Delta$	$\Delta$	$\Delta$	$\Delta$
SWI	(SP) - 2 $\Rightarrow$ SP; RTN <sub>H</sub> :RTN <sub>L</sub> $\Rightarrow$ M <sub>(SP):M<sub>(SP+1)</sub></sub> ; (SP) - 2 $\Rightarrow$ SP; (Y <sub>H</sub> :Y <sub>L</sub> ) $\Rightarrow$ M <sub>(SP):M<sub>(SP+1)</sub></sub> ; (SP) - 2 $\Rightarrow$ SP; (X <sub>H</sub> :X <sub>L</sub> ) $\Rightarrow$ M <sub>(SP):M<sub>(SP+1)</sub></sub> ; (SP) - 2 $\Rightarrow$ SP; (B:A) $\Rightarrow$ M <sub>(SP):M<sub>(SP+1)</sub></sub> ; (SP) - 1 $\Rightarrow$ SP; (CCR) $\Rightarrow$ M <sub>(SP)</sub> 1 $\Rightarrow$ I; (SWI Vector) $\Rightarrow$ PC  Software Interrupt	INH	3F	9	-	-	-	1	-	-	-	-
TAB	(A) $\Rightarrow$ B Transfer A to B	INH	18 0E	2	-	-	-	-	$\Delta$	$\Delta$	0	-
TAP	(A) $\Rightarrow$ CCR <i>Translates to TFR A, CCR</i>	INH	B7 02	1	$\Delta$	$\Downarrow$	$\Delta$	$\Delta$	$\Delta$	$\Delta$	$\Delta$	$\Delta$
TBA	(B) $\Rightarrow$ A Transfer B to A	INH	18 0F	2	-	-	-	-	$\Delta$	$\Delta$	0	-
TBEQ <i>cntr, rel</i>	If (cntr) = 0, then Branch; else Continue to next instruction  Test Counter and Branch if Zero (cntr = A, B, D, X, Y, or SP)	REL (9-bit)	04 lb rr	3	-	-	-	-	-	-	-	-
TBL <i>opr</i>	(M) + [(B) $\times$ ((M+1) - (M))] $\Rightarrow$ A 8-Bit Table Lookup and Interpolate  Initialize B, and index before TBL. <ea> points at first 8-bit table entry (M) and B is fractional part of lookup value.  (no indirect addressing modes allowed.)	IDX	18 3D xb	8	-	-	-	-	$\Delta$	$\Delta$	-	?
TBNE <i>cntr, rel</i>	If (cntr) not = 0, then Branch; else Continue to next instruction  Test Counter and Branch if Not Zero (cntr = A, B, D, X, Y, or SP)	REL (9-bit)	04 lb rr	3	-	-	-	-	-	-	-	-
TFR <i>r1, r2</i>	(r1) $\Rightarrow$ r2 or \$00:(r1) $\Rightarrow$ r2 or (r1[7:0]) $\Rightarrow$ r2  Transfer Register to Register r1 and r2 may be A, B, CCR, D, X, Y, or SP	INH	B7 eb	1	- or $\Delta$	- $\Downarrow$	-	-	-	-	-	-
TPA	(CCR) $\Rightarrow$ A <i>Translates to TFR CCR, A</i>	INH	B7 20	1	-	-	-	-	-	-	-	-

Table A-1 Instruction Set Summary (Continued)

Source Form	Operation	Addr. Mode	Machine Coding (hex)	~*	S	X	H	I	N	Z	V	C
TRAP	$(SP) - 2 \Rightarrow SP;$ $RTN_H; RTN_L \Rightarrow M_{(SP)}; M_{(SP+1)};$ $(SP) - 2 \Rightarrow SP; (Y_H; Y_L) \Rightarrow M_{(SP)}; M_{(SP+1)};$ $(SP) - 2 \Rightarrow SP; (X_H; X_L) \Rightarrow M_{(SP)}; M_{(SP+1)};$ $(SP) - 2 \Rightarrow SP; (B; A) \Rightarrow M_{(SP)}; M_{(SP+1)};$ $(SP) - 1 \Rightarrow SP; (CCR) \Rightarrow M_{(SP)}$ $1 \Rightarrow I; (TRAP \text{ Vector}) \Rightarrow PC$  Unimplemented opcode trap	INH	18 tn tn = \$30-\$39 or \$40-\$FF	10	-	-	-	1	-	-	-	-
TST opr	$(M) - 0$ Test Memory for Zero or Minus	EXT IDX IDX1 IDX2 [D,IDX] [IDX2]	F7 hh ll E7 xb E7 xb ff E7 xb ee ff E7 xb ee ff	3 3 3 4 6 6	-	-	-	-	$\Delta$	$\Delta$	0	0
TSTA	$(A) - 0$ Test A for Zero or Minus	INH	97	1	-	-	-	-	-	-	-	-
TSTB	$(B) - 0$ Test B for Zero or Minus	INH	D7	1	-	-	-	-	-	-	-	-
TSX	$(SP) \Rightarrow X$ <i>Translates to TFR SP,X</i>	INH	B7 75	1	-	-	-	-	-	-	-	-
TSY	$(SP) \Rightarrow Y$ <i>Translates to TFR SP,Y</i>	INH	B7 76	1	-	-	-	-	-	-	-	-
TXS	$(X) \Rightarrow SP$ <i>Translates to TFR X,SP</i>	INH	B7 57	1	-	-	-	-	-	-	-	-
TYS	$(Y) \Rightarrow SP$ <i>Translates to TFR Y,SP</i>	INH	B7 67	1	-	-	-	-	-	-	-	-
WAI	$(SP) - 2 \Rightarrow SP;$ $RTN_H; RTN_L \Rightarrow M_{(SP)}; M_{(SP+1)};$ $(SP) - 2 \Rightarrow SP; (Y_H; Y_L) \Rightarrow M_{(SP)}; M_{(SP+1)};$ $(SP) - 2 \Rightarrow SP; (X_H; X_L) \Rightarrow M_{(SP)}; M_{(SP+1)};$ $(SP) - 2 \Rightarrow SP; (B; A) \Rightarrow M_{(SP)}; M_{(SP+1)};$ $(SP) - 1 \Rightarrow SP; (CCR) \Rightarrow M_{(SP)};$  WAIT for interrupt	INH	3E	8** (in) + 5 (int)	-	-	-	-	-	-	-	-
WAV	$\sum_{i=1}^B S_i F_i \Rightarrow Y:D$ $\sum_{i=1}^B F_i \Rightarrow X$ Calculate Sum of Products and Sum of Weights for Weighted Average Calculation  Initialize B, X, and Y before WAV. B specifies number of elements. X points at first element in $S_i$ list. Y points at first element in $F_i$ list.  All $S_i$ and $F_i$ elements are 8-bits.  If interrupted, six extra bytes of stack used for intermediate values	Special	18 3C	8** per lable	-	-	?	-	?	$\Delta$	?	?



Table A-1 Instruction Set Summary (Continued)

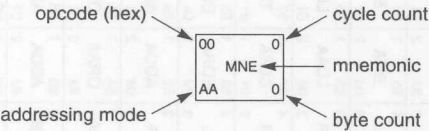
Source Form	Operation	Addr. Mode	Machine Coding (hex)	~*	S	X	H	I	N	Z	V	C
wavr	see WAV	Special	3C	**	-	-	?	-	?	Δ	?	?
pseudo-instruction	Resume executing an interrupted WAV instruction (recover intermediate results from stack rather than initializing them to zero)											
XGDX	(D) ⇔ (X) Translates to EXG D, X	INH	B7 C5	1	-	-	-	-	-	-	-	-
XGDY	(D) ⇔ (Y) Translates to EXG D, Y	INH	B7 C6	1	-	-	-	-	-	-	-	-

NOTES:

\*Each cycle (-) is typically 125 ns for an 8-MHz bus (16-MHz oscillator).

\*\*Refer to detailed instruction descriptions for additional information.

Key to Table A-2



Addressing mode abbreviations:

- DI — Direct
- EX — Extended
- ID — Indexed
- IH — Inherent
- IM — Immediate
- RL — Relative
- SP — Special

Cycle counts are for single-chip mode with 16-bit internal buses. Stack location (internal or external), external bus width, and operand alignment can affect actual execution time.



Table A-2 CPU12 Opcode Map (Sheet 1 of 2)

00	*5	10	1	20	3	30	3	40	1	50	1	60	3-6	70	4	80	1	90	3	A0	3-6	B0	3	C0	1	D0	3	E0	3-6	F0	3
BGND	IM	ANDCC	IM	BRA	RL	PULX	IH	NEGA	IH	NEGB	IH	NEG	ID	NEG	EX	SUBA	IM	SUBA	DI	SUBA	ID	SUBA	EX	SUBB	IM	SUBB	DI	SUBB	ID	SUBB	EX
01	5	11	11	21	1	31	3	41	1	51	1	61	3-6	71	4	81	1	91	3	A1	3-6	B1	3	C1	1	D1	3	E1	3-6	F1	3
MEM	IH	EDIV	IH	BRN	RL	PULY	IH	COMA	IH	COMB	IH	COM	ID	COM	EX	CMPA	IM	CMPA	DI	CMPA	ID	CMPA	EX	CMPB	IM	CMPB	DI	CMPB	ID	CMPB	EX
02	1	12	3	22	3/1	32	3	42	1	52	1	62	3-6	72	4	82	1	92	3	A2	3-6	B2	3	C2	1	D2	3	E2	3-6	F2	3
INY	IH	MUL	IH	BHI	RL	PULA	IH	INCA	IH	INCB	IH	INC	ID	INC	EX	SBCA	IM	SBCA	DI	SBCA	ID	SBCA	EX	SBCB	IM	SBCB	DI	SBCB	ID	SBCB	EX
03	1	13	3	23	3/1	33	3	43	1	53	1	63	3-6	73	4	83	2	93	3	A3	3-6	B3	3	C3	2	D3	3	E3	3-6	F3	3
DEY	IH	EMUL	IH	BLS	RL	PULB	IH	DECA	IH	DECB	IH	DEC	ID	DEC	EX	SUBD	IM	SUBD	DI	SUBD	ID	SUBD	EX	ADDD	IM	ADDD	DI	ADDD	ID	ADDD	EX
04	3	14	1	24	3/1	34	2	44	1	54	1	64	3-6	74	4	84	1	94	3	A4	3-6	B4	3	C4	1	D4	3	E4	3-6	F4	3
loop <sup>†</sup>	RL	ORCC	IM	BCC	RL	PSHX	IH	LSRA	IH	LSRB	IH	LSR	ID	LSR	EX	ANDA	IM	ANDA	DI	ANDA	ID	ANDA	EX	ANDB	IM	ANDB	DI	ANDB	ID	ANDB	EX
05	3-6	15	4-7	25	3/1	35	2	45	1	55	1	65	3-6	75	4	85	1	95	3	A5	3-6	B5	3	C5	1	D5	3	E5	3-6	F5	3
JMP	ID	JSR	ID	BCS	RL	PSHY	IH	ROLA	IH	ROLB	IH	ROL	ID	ROL	EX	BITA	IM	BITA	DI	BITA	ID	BITA	EX	BITB	IM	BITB	DI	BITB	ID	BITB	EX
06	3	16	4	26	3/1	36	2	46	1	56	1	66	3-6	76	4	86	1	96	3	A6	3-6	B6	3	C6	1	D6	3	E6	3-6	F6	3
JMP	EX	JSR	EX	BNE	RL	PSHA	IH	RORA	IH	RORB	IH	ROR	ID	ROR	EX	LDAA	IM	LDAA	DI	LDAA	ID	LDAA	EX	LDAB	IM	LDAB	DI	LDAB	ID	LDAB	EX
07	4	17	4	27	3/1	37	2	47	1	57	1	67	3-6	77	4	87	1	97	1	A7	1	B7	1	C7	1	D7	1	E7	3-6	F7	3
BSR	RL	JSR	DI	BEQ	RL	PSHB	IH	ASRA	IH	ASRB	IH	ASR	ID	ASR	EX	CLRA	IH	TSTA	IH	NOP	IH	TFR/EXG	IH	CLRB	IH	TSTB	IH	TST	ID	TST	EX
08	1	18	-	28	3/1	38	3	48	1	58	1	68	3-6	78	4	88	1	98	3	A8	3-6	B8	3	C8	1	D8	3	E8	3-6	F8	3
INX	IH	page 2	-	BVC	RL	PULC	IH	ASLA	IH	ASLB	IH	ASL	ID	ASL	EX	EORA	IM	EORA	DI	EORA	ID	EORA	EX	EORB	IM	EORB	DI	EORB	ID	EORB	EX
09	1	19	2	29	3/1	39	2	49	1	59	1	69	2-5	79	3	89	1	99	3	A9	3-6	B9	3	C9	1	D9	3	E9	3-6	F9	3
DEX	IH	LEAY	ID	BVS	RL	PSHC	IH	LSRD	IH	ASLD	IH	CLR	ID	CLR	EX	ADCA	IM	ADCA	DI	ADCA	ID	ADCA	EX	ADCB	IM	ADCB	DI	ADCB	ID	ADCB	EX
0A	6	1A	2	2A	3/1	3A	3	4A	8	5A	2	6A	2-5	7A	3	8A	1	9A	3	AA	3-6	BA	3	CA	1	DA	3	EA	3-6	FA	3
RTC	IH	LEAX	ID	BPL	RL	PULD	IH	CALL	EX	STAA	DI	STAA	ID	STAA	EX	ORAA	IM	ORAA	DI	ORAA	ID	ORAA	EX	ORAB	IM	ORAB	DI	ORAB	ID	ORAB	EX
0B	8	1B	2	2B	3/1	3B	2	4B	8-10	5B	2	6B	2-5	7B	3	8B	1	9B	3	AB	3-6	BB	3	CB	1	DB	3	EB	3-6	FB	3
RTI	IH	LEAS	ID	BMI	RL	PSHD	IH	CALL	ID	STAB	DI	STAB	ID	STAB	EX	ADDA	IM	ADDA	DI	ADDA	ID	ADDA	EX	ADDB	IM	ADDB	DI	ADDB	ID	ADDB	EX
0C	4-6	1C	4	2C	3/1	3C	*+9 wavr	4C	4	5C	2	6C	2-5	7C	3	8C	2	9C	3	AC	3-6	BC	3	CC	2	DC	3	EC	3-6	FC	3
BSET	ID	BSET	EX	BGE	RL	SP	1	BSET	DI	STD	DI	STD	ID	STD	EX	CPD	IM	CPD	DI	CPD	ID	CPD	EX	LDD	IM	LDD	DI	LDD	ID	LDD	EX
0D	4-6	1D	4	2D	3/1	3D	5	4D	4	5D	2	6D	2-5	7D	3	8D	2	9D	3	AD	3-6	BD	3	CD	2	DD	3	ED	3-6	FD	3
BCLR	ID	BCLR	EX	BLT	RL	IH	1	BCLR	DI	STY	DI	STY	ID	STY	EX	CPY	IM	CPY	DI	CPY	ID	CPY	EX	LDY	IM	LDY	DI	LDY	ID	LDY	EX
0E	4-8	1E	4	2E	3/1	3E	*8	4E	4	5E	2	6E	2-5	7E	3	8E	2	9E	3	AE	3-6	BE	3	CE	2	DE	3	EE	3-6	FE	3
BRSET	ID	BRSET	EX	BGT	RL	WAI	1	BRSET	DI	STX	DI	STX	ID	STX	EX	CPX	IM	CPX	DI	CPX	ID	CPX	EX	LDX	IM	LDX	DI	LDX	ID	LDX	EX
0F	4-8	1F	4	2F	3/1	3F	9	4F	4	5F	2	6F	2-5	7F	3	8F	2	9F	3	AF	3-6	BF	3	CF	2	DF	3	EF	3-6	FF	3
BRCLR	ID	BRCLR	EX	BLE	RL	SWI	1	BRCLR	DI	STS	DI	STS	ID	STS	EX	CPS	IM	CPS	DI	CPS	ID	CPS	EX	LDS	IM	LDS	DI	LDS	ID	LDS	EX

Table A-2 CPU12 Opcode Map (Sheet 2 of 2)

00	4	10	12	20	4	30	10	40	10	50	10	60	10	70	10	80	10	90	10	A0	10	B0	10	C0	10	D0	10	E0	10	F0	10
MOVW		IDIV		LBRA		TRAP		TRAP		TRAP		TRAP		TRAP		TRAP		TRAP		TRAP		TRAP		TRAP		TRAP		TRAP		TRAP	
IM-ID	5	IH	2	RL	4	IH	2	IH	2	IH	2	IH	2	IH	2	IH	2	IH	2	IH	2	IH	2	IH	2	IH	2	IH	2	IH	2
01	5	11	12	21	3	31	10	41	10	51	10	61	10	71	10	81	10	91	10	A1	10	B1	10	C1	10	D1	10	E1	10	F1	10
MOVW		FDIV		LBRN		TRAP		TRAP		TRAP		TRAP		TRAP		TRAP		TRAP		TRAP		TRAP		TRAP		TRAP		TRAP		TRAP	
EX-ID	5	IH	2	RL	4	IH	2	IH	2	IH	2	IH	2	IH	2	IH	2	IH	2	IH	2	IH	2	IH	2	IH	2	IH	2	IH	2
02	5	12	13	22	4/3	32	10	42	10	52	10	62	10	72	10	82	10	92	10	A2	10	B2	10	C2	10	D2	10	E2	10	F2	10
MOVW		EMACS		LBHI		TRAP		TRAP		TRAP		TRAP		TRAP		TRAP		TRAP		TRAP		TRAP		TRAP		TRAP		TRAP		TRAP	
ID-ID	4	SP	4	RL	4	IH	2	IH	2	IH	2	IH	2	IH	2	IH	2	IH	2	IH	2	IH	2	IH	2	IH	2	IH	2	IH	2
03	5	13	3	23	4/3	33	10	43	10	53	10	63	10	73	10	83	10	93	10	A3	10	B3	10	C3	10	D3	10	E3	10	F3	10
MOVW		EMULS		LBLS		TRAP		TRAP		TRAP		TRAP		TRAP		TRAP		TRAP		TRAP		TRAP		TRAP		TRAP		TRAP		TRAP	
IM-EX	6	IH	2	RL	4	IH	2	IH	2	IH	2	IH	2	IH	2	IH	2	IH	2	IH	2	IH	2	IH	2	IH	2	IH	2	IH	2
04	6	14	12	24	4/3	34	10	44	10	54	10	64	10	74	10	84	10	94	10	A4	10	B4	10	C4	10	D4	10	E4	10	F4	10
MOVW		EDIVS		LBCC		TRAP		TRAP		TRAP		TRAP		TRAP		TRAP		TRAP		TRAP		TRAP		TRAP		TRAP		TRAP		TRAP	
EX-EX	6	IH	2	RL	4	IH	2	IH	2	IH	2	IH	2	IH	2	IH	2	IH	2	IH	2	IH	2	IH	2	IH	2	IH	2	IH	2
05	5	15	12	25	4/3	35	10	45	10	55	10	65	10	75	10	85	10	95	10	A5	10	B5	10	C5	10	D5	10	E5	10	F5	10
MOVW		IDIVS		LBSC		TRAP		TRAP		TRAP		TRAP		TRAP		TRAP		TRAP		TRAP		TRAP		TRAP		TRAP		TRAP		TRAP	
ID-EX	5	IH	2	RL	4	IH	2	IH	2	IH	2	IH	2	IH	2	IH	2	IH	2	IH	2	IH	2	IH	2	IH	2	IH	2	IH	2
06	2	16	2	26	4/3	36	10	46	10	56	10	66	10	76	10	86	10	96	10	A6	10	B6	10	C6	10	D6	10	E6	10	F6	10
ABA		SBA		LBNE		TRAP		TRAP		TRAP		TRAP		TRAP		TRAP		TRAP		TRAP		TRAP		TRAP		TRAP		TRAP		TRAP	
IH	2	IH	2	RL	4	IH	2	IH	2	IH	2	IH	2	IH	2	IH	2	IH	2	IH	2	IH	2	IH	2	IH	2	IH	2	IH	2
07	3	17	2	27	4/3	37	10	47	10	57	10	67	10	77	10	87	10	97	10	A7	10	B7	10	C7	10	D7	10	E7	10	F7	10
DAA		CBA		LBEQ		TRAP		TRAP		TRAP		TRAP		TRAP		TRAP		TRAP		TRAP		TRAP		TRAP		TRAP		TRAP		TRAP	
IH	2	IH	2	RL	4	IH	2	IH	2	IH	2	IH	2	IH	2	IH	2	IH	2	IH	2	IH	2	IH	2	IH	2	IH	2	IH	2
08	4	18	4-7	28	4/3	38	10	48	10	58	10	68	10	78	10	88	10	98	10	A8	10	B8	10	C8	10	D8	10	E8	10	F8	10
MOVB		MAXA		LBVC		TRAP		TRAP		TRAP		TRAP		TRAP		TRAP		TRAP		TRAP		TRAP		TRAP		TRAP		TRAP		TRAP	
IM-ID	4	ID	3-5	RL	4	IH	2	IH	2	IH	2	IH	2	IH	2	IH	2	IH	2	IH	2	IH	2	IH	2	IH	2	IH	2	IH	2
09	5	19	4-7	29	4/3	39	10	49	10	59	10	69	10	79	10	89	10	99	10	A9	10	B9	10	C9	10	D9	10	E9	10	F9	10
MOVB		MINA		LBVS		TRAP		TRAP		TRAP		TRAP		TRAP		TRAP		TRAP		TRAP		TRAP		TRAP		TRAP		TRAP		TRAP	
EX-ID	5	ID	3-5	RL	4	IH	2	IH	2	IH	2	IH	2	IH	2	IH	2	IH	2	IH	2	IH	2	IH	2	IH	2	IH	2	IH	2
0A	5	1A	4-7	2A	4/3	3A	*3n	4A	10	5A	10	6A	10	7A	10	8A	10	9A	10	AA	10	BA	10	CA	10	DA	10	EA	10	FA	10
MOVB		EMAXD		LBPL		REV		TRAP		TRAP		TRAP		TRAP		TRAP		TRAP		TRAP		TRAP		TRAP		TRAP		TRAP		TRAP	
ID-ID	4	ID	3-5	RL	4	SP	2	IH	2	IH	2	IH	2	IH	2	IH	2	IH	2	IH	2	IH	2	IH	2	IH	2	IH	2	IH	2
0B	4	1B	4-7	2B	4/3	3B	*3n	4B	10	5B	10	6B	10	7B	10	8B	10	9B	10	AB	10	BB	10	CB	10	DB	10	EB	10	FB	10
MOVB		EMIND		LBMI		REVW		TRAP		TRAP		TRAP		TRAP		TRAP		TRAP		TRAP		TRAP		TRAP		TRAP		TRAP		TRAP	
IM-EX	5	ID	3-5	RL	4	SP	2	IH	2	IH	2	IH	2	IH	2	IH	2	IH	2	IH	2	IH	2	IH	2	IH	2	IH	2	IH	2
0C	6	1C	4-7	2C	4/3	3C	*8B	4C	10	5C	10	6C	10	7C	10	8C	10	9C	10	AC	10	BC	10	CC	10	DC	10	EC	10	FC	10
MOVB		MAXM		LBGE		WAV		TRAP		TRAP		TRAP		TRAP		TRAP		TRAP		TRAP		TRAP		TRAP		TRAP		TRAP		TRAP	
EX-EX	6	ID	3-5	RL	4	SP	2	IH	2	IH	2	IH	2	IH	2	IH	2	IH	2	IH	2	IH	2	IH	2	IH	2	IH	2	IH	2
0D	5	1D	4-7	2D	4/3	3D	8	4D	10	5D	10	6D	10	7D	10	8D	10	9D	10	AD	10	BD	10	CD	10	DD	10	ED	10	FD	10
MOVB		MINM		LBLT		TBL		TRAP		TRAP		TRAP		TRAP		TRAP		TRAP		TRAP		TRAP		TRAP		TRAP		TRAP		TRAP	
ID-EX	5	ID	3-5	RL	4	ID	3	IH	2	IH	2	IH	2	IH	2	IH	2	IH	2	IH	2	IH	2	IH	2	IH	2	IH	2	IH	2
0E	2	1E	4-7	2E	4/3	3E	*9+5	4E	10	5E	10	6E	10	7E	10	8E	10	9E	10	AE	10	BE	10	CE	10	DE	10	EE	10	FE	10
TAB		EMAXM		LBGT		STOP		TRAP		TRAP		TRAP		TRAP		TRAP		TRAP		TRAP		TRAP		TRAP		TRAP		TRAP		TRAP	
IH	2	ID	3-5	RL	4	IH	2	IH	2	IH	2	IH	2	IH	2	IH	2	IH	2	IH	2	IH	2	IH	2	IH	2	IH	2	IH	2
0F	2	1F	4-7	2F	4/3	3F	10	4F	10	5F	10	6F	10	7F	10	8F	10	9F	10	AF	10	BF	10	CF	10	DF	10	EF	10	FF	10
TBA		EMINM		LBLE		ETBL		TRAP		TRAP		TRAP		TRAP		TRAP		TRAP		TRAP		TRAP		TRAP		TRAP		TRAP		TRAP	
IH	2	ID	3-5	RL	4	ID	3	IH	2	IH	2	IH	2	IH	2	IH	2	IH	2	IH	2	IH	2	IH	2	IH	2	IH	2	IH	2

\* Refer to instruction glossary for more information.

‡ The opcode \$04 corresponds to one of the loop primitive instructions DBEQ, DBNE, IBEQ, IBNE, TBEQ, or TBNE.

Table A-3 Indexed Addressing Mode Postbyte Encoding (xb)

00 0,X 5b const	10 -16,X 5b const	20 1,+X pre-inc	30 1,X+ post-inc	40 0,Y 5b const	50 -16,Y 5b const	60 1,+Y pre-inc	70 1,Y+ post-inc	80 0,SP 5b const	90 -16,SP 5b const	A0 1,+SP pre-inc	B0 1,SP+ post-inc	C0 0,PC 5b const	D0 -16,PC 5b const	E0 n,X 9b const	F0 n,SP 9b const
01 1,X 5b const	11 -15,X 5b const	21 2,+X pre-inc	31 2,X+ post-inc	41 1,Y 5b const	51 -15,Y 5b const	61 2,+Y pre-inc	71 2,Y+ post-inc	81 1,SP 5b const	91 -15,SP 5b const	A1 2,+SP pre-inc	B1 2,SP+ post-inc	C1 1,PC 5b const	D1 -15,PC 5b const	E1 -n,X 9b const	F1 -n,SP 9b const
02 2,X 5b const	12 -14,X 5b const	22 3,+X pre-inc	32 3,X+ post-inc	42 2,Y 5b const	52 -14,Y 5b const	62 3,+Y pre-inc	72 3,Y+ post-inc	82 2,SP 5b const	92 -14,SP 5b const	A2 3,+SP pre-inc	B2 3,SP+ post-inc	C2 2,PC 5b const	D2 -14,PC 5b const	E2 n,X 16b const	F2 n,SP 16b const
03 3,X 5b const	13 -13,X 5b const	23 4,+X pre-inc	33 4,X+ post-inc	43 3,Y 5b const	53 -13,Y 5b const	63 4,+Y pre-inc	73 4,Y+ post-inc	83 3,SP 5b const	93 -13,SP 5b const	A3 4,+SP pre-inc	B3 4,SP+ post-inc	C3 3,PC 5b const	D3 -13,PC 5b const	E3 [n,X] 16b indir	F3 [n,SP] 16b indir
04 4,X 5b const	14 -12,X 5b const	24 5,+X pre-inc	34 5,X+ post-inc	44 4,Y 5b const	54 -12,Y 5b const	64 5,+Y pre-inc	74 5,Y+ post-inc	84 4,SP 5b const	94 -12,SP 5b const	A4 5,+SP pre-inc	B4 5,SP+ post-inc	C4 4,PC 5b const	D4 -12,PC 5b const	E4 A,X A offset	F4 A,SP A offset
05 5,X 5b const	15 -11,X 5b const	25 6,+X pre-inc	35 6,X+ post-inc	45 5,Y 5b const	55 -11,Y 5b const	65 6,+Y pre-inc	75 6,Y+ post-inc	85 5,SP 5b const	95 -11,SP 5b const	A5 6,+SP pre-inc	B5 6,SP+ post-inc	C5 5,PC 5b const	D5 -11,PC 5b const	E5 B,X B offset	F5 B,SP B offset
06 6,X 5b const	16 -10,X 5b const	26 7,+X pre-inc	36 7,X+ post-inc	46 6,Y 5b const	56 -10,Y 5b const	66 7,+Y pre-inc	76 7,Y+ post-inc	86 6,SP 5b const	96 -10,SP 5b const	A6 7,+SP pre-inc	B6 7,SP+ post-inc	C6 6,PC 5b const	D6 -10,PC 5b const	E6 D,X D offset	F6 D,SP D offset
07 7,X 5b const	17 -9,X 5b const	27 8,+X pre-inc	37 8,X+ post-inc	47 7,Y 5b const	57 -9,Y 5b const	67 8,+Y pre-inc	77 8,Y+ post-inc	87 7,SP 5b const	97 -9,SP 5b const	A7 8,+SP pre-inc	B7 8,SP+ post-inc	C7 7,PC 5b const	D7 -9,PC 5b const	E7 [D,X] D indirect	F7 [D,SP] D indirect
08 8,X 5b const	18 -8,X 5b const	28 8,-X pre-dec	38 8,X- post-dec	48 8,Y 5b const	58 -8,Y 5b const	68 8,-Y pre-dec	78 8,Y- post-dec	88 8,SP 5b const	98 -8,SP 5b const	A8 8,-SP pre-dec	B8 8,SP- post-dec	C8 8,PC 5b const	D8 -8,PC 5b const	E8 n,Y 9b const	F8 n,PC 9b const
09 9,X 5b const	19 -7,X 5b const	29 7,-X pre-dec	39 7,X- post-dec	49 9,Y 5b const	59 -7,Y 5b const	69 7,-Y pre-dec	79 7,Y- post-dec	89 9,SP 5b const	99 -7,SP 5b const	A9 7,-SP pre-dec	B9 7,SP- post-dec	C9 9,PC 5b const	D9 -7,PC 5b const	E9 -n,Y 9b const	F9 -n,PC 9b const
0A 10,X 5b const	1A -6,X 5b const	2A 6,-X pre-dec	3A 6,X- post-dec	4A 10,Y 5b const	5A -6,Y 5b const	6A 6,-Y pre-dec	7A 6,Y- post-dec	8A 10,SP 5b const	9A -6,SP 5b const	AA 6,-SP pre-dec	BA 6,SP- post-dec	CA 10,PC 5b const	DA -6,PC 5b const	EA n,Y 16b const	FA n,PC 16b const
0B 11,X 5b const	1B -5,X 5b const	2B 5,-X pre-dec	3B 5,X- post-dec	4B 11,Y 5b const	5B -5,Y 5b const	6B 5,-Y pre-dec	7B 5,Y- post-dec	8B 11,SP 5b const	9B -5,SP 5b const	AB 5,-SP pre-dec	BB 5,SP- post-dec	CB 11,PC 5b const	DB -5,PC 5b const	EB [n,Y] 16b indir	FB [n,PC] 16b indir
0C 12,X 5b const	1C -4,X 5b const	2C 4,-X pre-dec	3C 4,X- post-dec	4C 12,Y 5b const	5C -4,Y 5b const	6C 4,-Y pre-dec	7C 4,Y- post-dec	8C 12,SP 5b const	9C -4,SP 5b const	AC 4,-SP pre-dec	BC 4,SP- post-dec	CC 12,PC 5b const	DC -4,PC 5b const	EC A,Y A offset	FC A,PC A offset
0D 13,X 5b const	1D -3,X 5b const	2D 3,-X pre-dec	3D 3,X- post-dec	4D 13,Y 5b const	5D -3,Y 5b const	6D 3,-Y pre-dec	7D 3,Y- post-dec	8D 13,SP 5b const	9D -3,SP 5b const	AD 3,-SP pre-dec	BD 3,SP- post-dec	CD 13,PC 5b const	DD -3,PC 5b const	ED B,Y B offset	FD B,PC B offset
0E 14,X 5b const	1E -2,X 5b const	2E 2,-X pre-dec	3E 2,X- post-dec	4E 14,Y 5b const	5E -2,Y 5b const	6E 2,-Y pre-dec	7E 2,Y- post-dec	8E 14,SP 5b const	9E -2,SP 5b const	AE 2,-SP pre-dec	BE 2,SP- post-dec	CE 14,PC 5b const	DE -2,PC 5b const	EE D,Y D offset	FE D,PC D offset
0F 15,X 5b const	1F -1,X 5b const	2F 1,-X pre-dec	3F 1,X- post-dec	4F 15,Y 5b const	5F -1,Y 5b const	6F 1,-Y pre-dec	7F 1,Y- post-dec	8F 15,SP 5b const	9F -1,SP 5b const	AF 1,-SP pre-dec	BF 1,SP- post-dec	CF 15,PC 5b const	DF -1,PC 5b const	EF [D,Y] D indirect	FF [D,PC] D indirect

Table A-4 Transfer and Exchange Postbyte Encoding

TRANSFERS									
↓ LS	MS⇒	0	1	2	3	4	5	6	7
0		A ⇒ A	B ⇒ A	CCR ⇒ A	TMP3 <sub>L</sub> ⇒ A	B ⇒ A	X <sub>L</sub> ⇒ A	Y <sub>L</sub> ⇒ A	SP <sub>L</sub> ⇒ A
1		A ⇒ B	B ⇒ B	CCR ⇒ B	TMP3 <sub>L</sub> ⇒ B	B ⇒ B	X <sub>L</sub> ⇒ B	Y <sub>L</sub> ⇒ B	SP <sub>L</sub> ⇒ B
2		A ⇒ CCR	B ⇒ CCR	CCR ⇒ CCR	TMP3 <sub>L</sub> ⇒ CCR	B ⇒ CCR	X <sub>L</sub> ⇒ CCR	Y <sub>L</sub> ⇒ CCR	SP <sub>L</sub> ⇒ CCR
3		sex:A ⇒ TMP2	sex:B ⇒ TMP2	sex:CCR ⇒ TMP2	TMP3 ⇒ TMP2	D ⇒ TMP2	X ⇒ TMP2	Y ⇒ TMP2	SP ⇒ TMP2
4		sex:A ⇒ D SEX A,D	sex:B ⇒ D SEX B,D	sex:CCR ⇒ D SEX CCR,D	TMP3 ⇒ D	D ⇒ D	X ⇒ D	Y ⇒ D	SP ⇒ D
5		sex:A ⇒ X SEX A,X	sex:B ⇒ X SEX B,X	sex:CCR ⇒ X SEX CCR,X	TMP3 ⇒ X	D ⇒ X	X ⇒ X	Y ⇒ X	SP ⇒ X
6		sex:A ⇒ Y SEX A,Y	sex:B ⇒ Y SEX B,Y	sex:CCR ⇒ Y SEX CCR,Y	TMP3 ⇒ Y	D ⇒ Y	X ⇒ Y	Y ⇒ Y	SP ⇒ Y
7		sex:A ⇒ SP SEX A,SP	sex:B ⇒ SP SEX B,SP	sex:CCR ⇒ SP SEX CCR,SP	TMP3 ⇒ SP	D ⇒ SP	X ⇒ SP	Y ⇒ SP	SP ⇒ SP
EXCHANGES									
↓ LS	MS⇒	8	9	A	B	C	D	E	F
0		A ⇔ A	B ⇔ A	CCR ⇔ A	TMP3 <sub>L</sub> ⇔ A \$00:A ⇒ TMP3	B ⇔ A A ⇔ B	X <sub>L</sub> ⇔ A \$00:A ⇒ X	Y <sub>L</sub> ⇔ A \$00:A ⇒ Y	SP <sub>L</sub> ⇔ A \$00:A ⇒ SP
1		A ⇔ B	B ⇔ B	CCR ⇔ B	TMP3 <sub>L</sub> ⇔ B \$FF:B ⇒ TMP3	B ⇔ B \$FF ⇒ A	X <sub>L</sub> ⇔ B \$FF:B ⇒ X	Y <sub>L</sub> ⇔ B \$FF:B ⇒ Y	SP <sub>L</sub> ⇔ B \$FF:B ⇒ SP
2		A ⇔ CCR	B ⇔ CCR	CCR ⇔ CCR	TMP3 <sub>L</sub> ⇔ CCR \$FF:CCR ⇒ TMP3	B ⇔ CCR \$FF:CCR ⇒ D	X <sub>L</sub> ⇔ CCR \$FF:CCR ⇒ X	Y <sub>L</sub> ⇔ CCR \$FF:CCR ⇒ Y	SP <sub>L</sub> ⇔ CCR \$FF:CCR ⇒ SP
3		\$00:A ⇒ TMP2 TMP2 <sub>L</sub> ⇒ A	\$00:B ⇒ TMP2 TMP2 <sub>L</sub> ⇒ B	\$00:CCR ⇒ TMP2 TMP2 <sub>L</sub> ⇒ CCR	TMP3 ⇔ TMP2	D ⇔ TMP2	X ⇔ TMP2	Y ⇔ TMP2	SP ⇔ TMP2
4		\$00:A ⇒ D	\$00:B ⇒ D	\$00:CCR ⇒ D B ⇒ CCR	TMP3 ⇔ D	D ⇔ D	X ⇔ D	Y ⇔ D	SP ⇔ D
5		\$00:A ⇒ X X <sub>L</sub> ⇒ A	\$00:B ⇒ X X <sub>L</sub> ⇒ B	\$00:CCR ⇒ X X <sub>L</sub> ⇒ CCR	TMP3 ⇔ X	D ⇔ X	X ⇔ X	Y ⇔ X	SP ⇔ X
6		\$00:A ⇒ Y Y <sub>L</sub> ⇒ A	\$00:B ⇒ Y Y <sub>L</sub> ⇒ B	\$00:CCR ⇒ Y Y <sub>L</sub> ⇒ CCR	TMP3 ⇔ Y	D ⇔ Y	X ⇔ Y	Y ⇔ Y	SP ⇔ Y
7		\$00:A ⇒ SP SP <sub>L</sub> ⇒ A	\$00:B ⇒ SP SP <sub>L</sub> ⇒ B	\$00:CCR ⇒ SP SP <sub>L</sub> ⇒ CCR	TMP3 ⇔ SP	D ⇔ SP	X ⇔ SP	Y ⇔ SP	SP ⇔ SP



SKC



## APPENDIX B

### M68HC11 TO M68HC12 UPGRADE PATH

This appendix discusses similarities and differences between the CPU12 and the M68HC11 CPU. In general, the CPU12 is a proper superset of the M68HC11. Significant changes have been made to improve the efficiency and capabilities of the CPU without giving up compatibility and familiarity for the large community of M68HC11 programmers.

#### B.1 CPU12 Design Goals

The primary goals of the CPU12 design were:

- ABSOLUTE source code compatibility with the M68HC11
- Same programming model
- Same stacking operations
- Upgrade to 16-bit architecture
- Eliminate extra byte/extra cycle penalty for using index register Y
- Improve performance
- Improve compatibility with high level languages

#### B.2 Source Code Compatibility

**Every M68HC11 instruction mnemonic and source code statement can be assembled directly with a CPU12 assembler with no modifications.**

The CPU12 supports all M68HC11 addressing modes and includes several new variations of indexed addressing mode. CPU12 instructions affect condition code bits in the same way as M68HC11 instructions.

CPU12 object code is similar to but not identical to M68HC11 object code. Some primary objectives, such as the elimination of the penalty for using Y, could not be achieved without object code differences. While the object code has been changed, the majority of the opcodes are identical to those of the M6800, which was developed more than 20 years earlier.

The CPU12 assembler automatically translates a few M68HC11 instruction mnemonics into functionally equivalent CPU12 instructions. For example, the CPU12 does not have an increment stack pointer (INS) instruction, so the INS mnemonic is translated to LEAS 1,S. The CPU12 does provide single-byte DEX, DEY, INX, and INY instructions because the LEAX and LEAY instructions do not affect the condition codes, while the M68HC11 instructions update the Z bit according to the result of the decrement or increment.

**Table B-1** shows M68HC11 instruction mnemonics that are automatically translated into equivalent CPU12 instructions. This translation is performed by the assembler so there is no need to modify an old M68HC11 program in order to assemble it for the CPU12. In fact, the M68HC11 mnemonics can be used in new CPU12 programs.



CRUZE. In fact, the MEBHCT1 assembler can be used in new CRUZE programs. There is no need to modify an old MEBHCT1 program in order to assemble it for the new CRUZE. Table B-1 shows MEBHCT1 instruction mnemonics that are automatically translated into equivalent CRUZE instructions. This translation is performed by the assembler so there is no need to modify an old MEBHCT1 program in order to assemble it for the new CRUZE.

ment or increment.

while the MEBHCT1 instructions update the Z bit according to the result of the operation because the LEAX and LEAY instructions do not affect the condition codes. LEAS 1. The CRUZE does provide single-byte DEC, DEC INX, and INY instructions. The CRUZE assembler automatically translates a few MEBHCT1 instruction mnemonics into functionally equivalent CRUZE instructions. For example, the CRUZE does not have an increment stack pointer (INSP) instruction, so the INSP mnemonic is translated to LEAS 1. The CRUZE does provide single-byte DEC, DEC INX, and INY instructions because the LEAX and LEAY instructions do not affect the condition codes.

more than 50 years earlier.

the majority of the opcodes are identical to those of the ME800, which was developed achieved without object code differences. While the object code has been changed many objectives, such as the elimination of the penalty for using Y could not be CRUZE object code is similar to but not identical to MEBHCT1 object code. Some differences are as follows:

named directly with a CRUZE assembler with no modifications.

## B.2 Source Code Compatibility

g/c

- Improve compatibility with high level languages
- Improve performance
- Eliminate extra byte/word cycle penalty for using index register Y
- Upgrade to 16-bit architecture
- Same addressing operations
- Same programming model
- ABSOLUTE source code compatibility with the MEBHCT1

The primary goals of the CRUZE design were:

## B.1 CRUZE Design Goals

programs.

## MEBHCT1 TO MEBHCT2 UPGRADE PATH

### APPENDIX B

S/C

#### **B.4.1 Bus Structures**

The CPU12 is a 16-bit processor with 16-bit data paths. Typical M68HC12 devices have internal and external 16-bit data paths, but some derivatives incorporate operating modes that allow for an 8-bit data bus, so that a system can be built with low-cost 8-bit program memory. M68HC12 MCUs include an on-chip integration module that manages the external bus interface. When the CPU makes a 16-bit access to a resource that is served by an 8-bit bus, the integration module performs two 8-bit accesses, freezes the CPU clocks for part of the sequence, and assembles the data into a 16-bit word. As far as the CPU is concerned, there is no difference between this access and a 16-bit access to an internal resource via the 16-bit data bus. This is similar to the way an M68HC11 can stretch clock cycles to accommodate slow peripherals.

#### **B.4.2 Instruction Queue**

The CPU12 has a two-word instruction queue and a 16-bit holding buffer, which sometimes acts as a third word for queueing program information. All program information is fetched from memory as aligned 16-bit words, even though there is no requirement for instructions to begin or end on even word boundaries. There is no penalty for misaligned instructions. If a program begins on an odd boundary (if the reset vector is an odd address), program information is fetched to fill the instruction queue, beginning with the aligned word at the next address below the misaligned reset vector. The instruction queue logic starts execution with the opcode in the low order half of this word.

The instruction queue causes three bytes of program information (starting with the instruction opcode) to be directly available to the CPU at the beginning of every instruction. As it executes, each instruction performs enough additional program fetches to refill the space it took up in the queue. Alignment information is maintained by the logic in the instruction queue. The CPU provides signals that tell the queue logic when to advance a word of program information, and when to toggle the alignment status.

The CPU is not aware of instruction alignment. The queue logic includes a multiplexer that sorts out the information in the queue to present the opcode and the next two bytes of information as CPU inputs. The multiplexer determines whether the opcode is in the even or odd half of the word at the head of the queue. Alignment status is also available to the ALU for address calculations. The execution sequence for all instructions is independent of the alignment of the instruction.

The only situation where alignment can affect the number of cycles an instruction takes occurs in devices that have a narrow (8-bit) external data bus, and is related to optional program fetch cycles (O type cycles). O cycles are always performed, but serve different purposes determined by instruction size and alignment.

Each instruction includes one program fetch cycle for every two bytes of object code. Instructions with an odd number of bytes can use an O cycle to fetch an extra word of object code. If the queue is aligned at the start of an instruction with an odd byte count, the last byte of object code shares a queue word with the opcode of the next instruction. Since this word holds part of the next instruction, the queue cannot advance after the odd byte executes, or the first byte of the next instruction would be

lost. In this case, the O cycle appears as a free cycle since the queue is not ready to accept the next word of program information. If this same instruction had been misaligned, the queue would be ready to advance and the O cycle would be used to perform a program word fetch.

In a single-chip system or in a system with the program in 16-bit memory, both the free cycle and the program fetch cycle take one bus cycle. In a system with the program in an external 8-bit memory, the O cycle takes one bus cycle when it appears as a free cycle, but it takes two bus cycles when used to perform a program fetch. In this case, the on-chip integration module freezes the CPU clocks long enough to perform the cycle as two smaller accesses. The CPU handles only 16-bit data, and is not aware that the 16-bit program access is split into two 8-bit accesses.

In order to allow development systems to track events in the CPU12 instruction queue, two status signals (IPIPE[1:0]) provide information about data movement in the queue and about the start of instruction execution. A development system can use this information along with address and data information to externally reconstruct the queue. This representation of the queue can also track both the data and address buses.

#### **B.4.3 Stack Function**

Both the M68HC11 and the CPU12 stack nine bytes for interrupts. Since this is an odd number of bytes, there is no practical way to assure that the stack will stay aligned. To assure that instructions take a fixed number of cycles regardless of stack alignment, the internal RAM in M68HC12 MCUs is designed to allow single cycle 16-bit accesses to misaligned addresses. As long as the stack is located in this special RAM, stacking and unstacking operations take the same amount of execution time, regardless of stack alignment. If the stack is located in an external 16-bit RAM, a PSHX instruction can take two or three cycles depending upon the alignment of the stack. This extra access time is transparent to the CPU because the integration module freezes the CPU clocks while it performs the extra 8-bit bus cycle required for a misaligned stack operation.

The CPU12 has a "last-used" stack rather than a "next-available" stack like the M68HC11 CPU. That is, the stack pointer points to the last 16-bit stack address used, rather than to the address of the next available stack location. This generally has very little effect, because it is very unusual to access stacked information using absolute addressing. The change allows a 16-bit word of data to be removed from the stack without changing the value of the SP twice.

To illustrate, consider the operation of a PULX instruction. With the next-available M68HC11 stack, if the SP = \$01F0 when execution begins, the sequence of operations is: SP = SP + 1; load X from \$01F1:01F2; SP = SP + 1; and the SP ends up at \$01F2. With the last-used CPU12 stack, if the SP = \$01F0 when execution begins, the sequence is: load X from \$01F0:01F1; SP = SP + 2; and the SP again ends up at \$01F2. The second sequence requires one less stack pointer adjustment.

lost in this case, the O cycle appears as a free cycle since the queue is not ready to accept the next word of program information. If the same instruction had been misaligned, the queue would be ready to advance and the O cycle would be used to perform a program word latch.

In a single-chip system or in a system with the program in 16-bit memory, both the free cycle and the program latch cycle take one bus cycle. In a system with the program in an external 8-bit memory, the O cycle takes one bus cycle when it appears as a free cycle, but it takes two bus cycles when used to perform a program latch. In this case, the on-chip integration module freezes the CPU clock long enough to perform the cycle as two smaller accesses. The CPU handles only 16-bit data, and is not aware that the 16-bit program access is split into two 8-bit accesses.

In order to allow development systems to track events in the CPU12 instruction queue, two status signals (PVS/O) provide information about data movement in the queue and about the start of instruction execution. A development system can use this information along with address and data information to externally reconstruct the queue. This representation of the queue can also track both the data and address buses.

### B.4.3 Stack Function

Both the M88HC11 and the CPU12 stack nine bytes for interrupts. Since this is an odd number of bytes, there is no practical way to assure that the stack will stay aligned. To assure that instructions take a 16-bit number of cycles regardless of stack alignment, the internal RAM in M88HC11/M88HC12 is designed to allow single cycle 16-bit accesses to misaligned addresses. As the stack is located in this special RAM, stacking and unstacking operations take a constant amount of execution time, regardless of stack alignment. If the stack is located in an external 16-bit RAM, a PUSH instruction can take two or three cycles depending upon the alignment of the stack. This extra access time is transparent to the CPU because the integration module freezes the CPU clock while it performs the extra 8-bit bus cycles required for a misaligned stack operation.

The CPU12 has a "last-used" stack rather than a "next-available" stack like the M88HC11 CPU. That is, the stack pointer points to the last 16-bit stack address used, rather than to the address of the next available stack location. This generally has very little effect, because it is very unusual to access stacked information using absolute addressing. The change allows a 16-bit word of data to be removed from the stack without changing the value of the SP twice.

To illustrate, consider the operation of a PULX instruction. With the next-available M88HC11 stack, if the SP = \$01F0 when execution begins, the sequence of operations is: SP = SP + 1; load X from \$01F0:01F1; SP = SP + 1; and the SP ends up at \$01F2. With the last-used CPU12 stack, if the SP = \$01F0 when execution begins, the sequence is: load X from \$01F0:01F1; SP = SP + 2; and the SP again ends up at \$01F2. The second sequence requires one less stack pointer adjustment.



### B.5.2 Auto-Increment Indexing

The CPU12 provides greatly enhanced auto increment and decrement modes of indexed addressing. In the CPU12, the index modification may be specified for before the index is used (pre-), or after the index is used (post-), and the index can be incremented or decremented by any amount from one to eight, independent of the size of the operand that was accessed. X, Y, and SP can be used as the index reference, but this mode does not allow PC to be the index reference (this would interfere with proper program execution).

This addressing mode can be used to implement a software stack structure, or to manipulate data structures in lists or tables, rather than manipulating bytes or words of data. Anywhere an M68HC11 program has an increment or decrement index register operation near an indexed mode instruction, the increment or decrement operation can be combined with the indexed instruction with no cost in object code size, as shown in the following code comparison.

18 A6 00	LDAA 0,Y	A6 71	LDAA 2,Y+
18 08	INY		
18 08	INY		

The M68HC11 object code requires seven bytes, while the CPU12 requires only two bytes to accomplish the same functions. Three bytes of M68HC11 code were due to the page prebyte for each Y-related instruction (\$18). CPU12 post-increment indexing capability allowed the two INY instructions to be absorbed into the LDAA indexed instruction. The replacement code is not identical to the original three instruction sequence because the Z condition code bit is affected by the M68HC11 INY instructions, while the Z bit in the CPU12 would be determined by the value loaded into A.

### B.5.3 Accumulator Offset Indexing

This indexed addressing variation allows the programmer to use either an 8-bit accumulator (A or B), or the 16-bit D accumulator as the offset for indexed addressing. This allows for a program-generated offset, which is more difficult to achieve in the M68HC11. The following code compares the M68HC11 and CPU12 operations.

C6 05	LDAB #\$5	[2]	C6 05	LDAB #\$5	[1]
CE 10 00	LOOP LDX #\$1000	[3]	CE 10 00	LDX #\$1000	[2]
3A	ABX	[3]	A6 E5	LOOP LDAA B,X	[3]
A6 00	LDAA 0,X	[4]			
			04 31 FB	DBNE B,LOOP	[3]
5A	DECB	[2]			
26 F7	BNE LOOP	[3]			

The CPU12 object code is only one byte smaller, but the LDX # instruction is outside the loop. It is not necessary to reload the base address in the index register on each pass through the loop because the LDAA B,X instruction does not alter the index register. This reduces the loop execution time from 15 cycles to six cycles. This reduction, combined with the 8-MHz bus speed of the M68HC12 family, can have significant effects.

#### B.5.4 Indirect Indexing

The CPU12 allows some forms of indexed indirect addressing where the instruction points to a location in memory where the address of the operand is stored. This is an extra level of indirection compared to ordinary indexed addressing. The two forms of indexed indirect addressing are 16-bit constant offset indexed indirect and D accumulator indexed indirect. The reference index register can be X, Y, SP, or PC as in other CPU12 indexed addressing modes. PC-relative indirect addressing is one of the more common uses of indexed indirect addressing. The indirect variations of indexed addressing help in the implementation of pointers. D accumulator indexed indirect addressing can be used to implement a runtime computed GOTO function. Indirect addressing is also useful in high level language compilers. For instance, PC-relative indirect indexing can be used to efficiently implement some C case statements.

#### B.6 Improved Performance

The CPU12 improves on M68HC11 performance in several ways. M68HC12 devices are designed using sub-micron design rules, and fabricated using advanced semiconductor processing, the same methods used to manufacture the M68HC16 and M68300 families of modular microcontrollers. M68HC12 devices have a base bus speed of eight MHz, and are designed to operate over a wide range of supply voltages. The 16-bit wide architecture also increases performance. Beyond these obvious improvements, the CPU12 uses a reduced number of cycles for many of its instructions, and a 20-bit ALU makes certain CPU12 math operations much faster.

##### B.6.1 Reduced Cycle Counts

No M68HC11 instruction takes less than two cycles, but the CPU12 has more than 50 opcodes that take only one cycle. Some of the reduction comes from the instruction queue, which assures that several program bytes are available at the start of each instruction. Other cycle reductions occur because the CPU12 can fetch 16 bits of information at a time, rather than eight bits at a time.

##### B.6.2 Fast Math

The CPU12 has some of the fastest math ever designed into a Motorola general-purpose MCU. Much of the speed is due to a 20-bit ALU that can perform two smaller operations simultaneously. The ALU can also perform two operations in a single bus cycle in certain cases. **Table B-3** compares the speed of CPU12 and M68HC11 math instructions. The CPU12 requires fewer cycles to perform an operation, and the cycle time is half that of the M68HC11.



## B.2.4 Indirect Indexing

The CPU12 allows some forms of indirect addressing when the instruction points to a location in memory where the address of the operand is stored. This is an extra level of indirection compared to ordinary indexed addressing. The two forms of indexed indirect addressing are 16-bit constant offset indirect and D accumulator-indirect indirect. The reference index register can be X, Y, SP, or PC as in other CPU12 indexed addressing modes. PC-relative indirect addressing is one of the more common uses of indirect indirect addressing. The indirect versions of indexed addressing help in the implementation of pointers. D accumulator indirect addressing can be used to implement a runtime computed GOTO function. Indirect addressing is also useful in high-level language compilers. For instance, PC-relative indirect indexing can be used to efficiently implement some G-code statements.

## B.2 Improved Performance

The CPU12 improves on M68HC11 performance in several ways. M68HC12 devices are designed using sub-micron design rules, and fabricated using advanced semiconductor processing, the same methods used to manufacture the M68HC16 and M68300 families of modular microcontrollers. M68HC12 devices have a base bus speed of eight MHz, and are designed to operate over a wide range of supply voltages. The 16-bit wide architecture also increases performance. Beyond these obvious improvements, the CPU12 uses a reduced number of cycles for many of its instructions, and a 30-bit ALU makes certain CPU12 math operations much faster.

## B.2.1 Reduced Cycle Counts

In M68HC11 instruction tables less than two cycles. CPU12 has more than 50 opcodes that take only one cycle. Some of the reduction comes from the instruction queue, which assures that several program bytes are available at the start of each instruction. Other cycle reductions occur because the CPU12 can fetch 16 bits of information at a time, rather than eight bits at a time.

## B.2.2 Fast Math

The CPU12 has some of the fastest math ever designed into a Motorola general-purpose MCU. Much of this speed is due to a 30-bit ALU that can perform two or three operations simultaneously. The ALU can also perform two operations in a single bus cycle in certain cases. Table B-3 compares the speed of CPU12 and M68HC11 math instructions. The CPU12 requires fewer cycles to perform an operation, and the cycle time is half that of the M68HC11.



**Table B-4 New M68HC12 Instructions (Continued)**

Mnemonic	Addressing Modes	Brief Functional Description
LBCC	Relative	Long Branch if Carry Clear (Same as LBHS)
LBCS	Relative	Long Branch if Carry Set (Same as LBLO)
LBEQ	Relative	Long Branch if Equal (Z=1)
LBGE	Relative	Long Branch if Greater than or Equal to Zero
LBGT	Relative	Long Branch if Greater than Zero
LBHI	Relative	Long Branch if Higher
LBHS	Relative	Long Branch if Higher or Same (Same as LBCC)
LBLE	Relative	Long Branch if Less than or Equal to Zero
LBLO	Relative	Long Branch if Lower (Same as LBCS)
LBS	Relative	Long Branch if Lower or Same
LBLT	Relative	Long Branch if Less than Zero
LBMI	Relative	Long Branch if Minus
LBNE	Relative	Long Branch if Not Equal to Zero
LBPL	Relative	Long Branch if Plus
LBRA	Relative	Long Branch Always
LBRN	Relative	Long Branch Never
LBVC	Relative	Long Branch if Overflow Clear
LBVS	Relative	Long Branch if Overflow Set
LEAS	Indexed	Load Stack Pointer with Effective Address
LEAX	Indexed	Load X Index Register with Effective Address
LEAY	Indexed	Load Y Index Register with Effective Address
MAXA	Indexed	Maximum of Two Unsigned 8-Bit Values
MAXM	Indexed	Maximum of Two Unsigned 8-Bit Values
MEM	Special	Determine Grade of Fuzzy Membership
MINA	Indexed	Minimum of Two Unsigned 8-Bit Values
MINM	Indexed	Minimum of Two Unsigned 8-Bit Values
MOVB(W)	Combinations of Immediate, Extended, and Indexed	Move Data from One Memory Location to Another
ORCC	Immediate	OR CCR with Mask (replaces SEC, SEI, and SEV)
PSHC	Inherent	Push CCR onto Stack
PSHD	Inherent	Push Double Accumulator onto Stack
PULC	Inherent	Pull CCR Contents from Stack
PULD	Inherent	Pull Double Accumulator from Stack
REV	Special	Fuzzy Logic Rule Evaluation
REVV	Special	Fuzzy Logic Rule Evaluation with Weights
RTC	Inherent	Restore Program Page and Return Address from Stack Used with CALL Instruction, Allows Easy Access to >64-Kbyte Space
SEX	Inherent	Sign Extend 8-bit Register into 16-bit Register
TBEQ	Relative	Test and Branch if Equal to Zero (Looping Primitive)
TBL	Inherent	Table Lookup and Interpolate (8-bit Entries)
TBNE	Relative	Test Register and Branch if Not Equal to Zero (Looping Primitive)
TFR	Inherent	Transfer Register Contents to Another Register
WAV	Special	Weighted Average (Fuzzy Logic Support)

### B.7.1 Memory-to-Memory Moves

The CPU12 has both 8- and 16-bit variations of memory-to-memory move instructions. The source address can be specified with immediate, extended, or indexed addressing modes. The destination address can be specified by extended or indexed addressing mode. The indexed addressing mode for move instructions is limited to modes that require no extension bytes (9- and 16-bit constant offsets are not allowed), and indirect indexing is not allowed for moves. This leaves a 5-bit signed constant offset, accumulator offsets, and the automatic increment/decrement modes. The following simple loop is a block move routine capable of moving up to 256 words of information from one memory area to another.

```
LOOP  MOVW  2,X+ , 2,Y+ ;move a word and update pointers
      DBNE  B,LOOP      ;repeat B times
```

The move immediate to extended is a convenient way to initialize a register without using an accumulator or affecting condition codes.

### B.7.2 Universal Transfer and Exchange

The M68HC11 has only eight transfer instructions and two exchange instructions. The CPU12 has a universal transfer/exchange instruction that can be used to transfer or exchange data between any two CPU registers. The operation is obvious when the two registers are the same size, but some of the other combinations provide very useful results. For example when an 8-bit register is transferred to a 16-bit register, a sign-extend operation is performed. Other combinations can be used to perform a zero-extend operation.

These instructions are used often in CPU12 assembly language programs. Transfers can be used to make extra copies of data in another register, and exchanges can be used to temporarily save data during a call to a routine that expects data in a specific register. This is sometimes faster and produces more compact object code than saving data to memory with pushes or stores.

### B.7.3 Loop Construct

The CPU12 instruction set includes a new family of six loop primitive instructions. These instructions decrement, increment, or test a loop count in a CPU register and then branch based on a zero or non-zero test result. The CPU registers that can be used for the loop count are A, B, D, X, Y, or SP. The branch range is a 9-bit signed value (-512 to +511) which gives these instructions twice the range of a short branch instruction.

### B.7.4 Long Branches

All of the branch instructions from the M68HC11 are also available with 16-bit offsets which allows them to reach any location in the 64-Kbyte address space.

### B.7.4 Long Branches

All of the branch instructions from the MSHC1 are also available with 16-bit offsets which allow them to reach any location in the 64-Kbyte address space.

### B.7.3 Loop Construct

The CPU12 instruction set includes a new family of six loop primitive instructions. These instructions decrement, increment, or test a loop count in a CPU register and then branch based on a zero or non-zero test result. The CPU registers that can be used for the loop count are A, B, D, X, Y, or SR. The branch range is a 9-bit signed value (-512 to +511) which gives these instructions twice the range of a short branch instruction.

These instructions are used often in CPU12 assembly language programs. Transfers can be used to make extra copies of data in another register and exchanges can be used to interchange save data during a call to a routine that expects data in a specific register. This is sometimes faster and produces more compact object code than saving data to memory with pushes or stores.

Two registers are the same size, but some of the other conditions provide very useful results. For example when an 8-bit register is transferred to a 16-bit register a sign-extend operation is performed. Other conditions can be used to perform a zero-extend operation.

The MSHC1 has only eight transfer instructions and two exchange instructions. The CPU12 has a universal transfer/exchange instruction that can be used to transfer or exchange data between any two CPU registers. The operation is obvious when the two registers are the same size, but some of the other conditions provide very useful results. For example when an 8-bit register is transferred to a 16-bit register a sign-extend operation is performed. Other conditions can be used to perform a zero-extend operation.

### B.7.2 Universal Transfer and Exchange

The move immediate is extended to a convenient way to initialize a register without using an immediate or affecting condition codes.

MOVE 0,1000 ; Move 0 to R10  
MOVE 0,1000 ; Move 0 to R10

### B.7.1 Memory-to-Memory Moves

The CPU12 has both 8- and 16-bit versions of memory-to-memory move instructions. The source address can be specified with immediate, extended, or indirect addressing modes. The destination address can be specified by extended or indirect addressing mode. The indirect addressing mode for move instructions is limited to modes that require no extension bytes (B- and 16-bit constant offsets are not allowed), and indirect indexing is not allowed for moves. This leaves a 2-bit signed constant offset, accumulator offsets, and the automatic increment/decrement modes. The following simple loop is a block move routine capable of moving up to 256 words of information from one memory area to another.

## B.3.11 Support for Memory Expansion

Bank switching is a common method of expanding memory beyond the 64-Kbyte limit of a CPU with a 64-Kbyte address space, but there are some known difficulties associated with bank switching. One problem is that interrupts cannot take place during the bank switching operation. This increases worst case interrupt latency and requires extra programming space and execution time.

Some M68HC12 variants include a built-in bank switching scheme that eliminates many of the problems associated with external switching logic. The CPU12 includes CALL and return from call (RTC) instructions that manage the switch to the bank-switching system. These instructions are analogous to the JSR and RTS instructions, except that the bank page number is saved and restored automatically during execution. Since the page change operation is part of an uninterpretable instruction, many of the difficulties associated with bank switching are eliminated. On M68HC12 devices with expanded memory, external bank numbers are specified by on-chip control registers. Since the addresses of these control registers may not be the same in all M68HC12 derivatives, the CPU12 has a dedicated control line to the on-chip instruction module that indicates when a memory-expansion register is being read or written. This allows the CPU to access the PRAGE register without knowing the register address.

The indexed indirect variants of the CALL instruction access the address of the called routine and the destination page value. In the indexed indirect mode, the value of the CALL instruction, the destination page value is provided as memory data in the instruction offset code. CALL and RTC execute correctly in the normal 64-Kbyte address space, thus providing for portable code.

S/C

### **B.7.11 Support for Memory Expansion**

Bank switching is a common method of expanding memory beyond the 64-Kbyte limit of a CPU with a 64-Kbyte address space, but there are some known difficulties associated with bank switching. One problem is that interrupts cannot take place during the bank switching operation. This increases worst case interrupt latency and requires extra programming space and execution time.

Some M68HC12 variants include a built-in bank switching scheme that eliminates many of the problems associated with external switching logic. The CPU12 includes CALL and return from call (RTC) instructions that manage the interface to the bank-switching system. These instructions are analogous to the JSR and RTS instructions, except that the bank page number is saved and restored automatically during execution. Since the page change operation is part of an uninterruptable instruction, many of the difficulties associated with bank switching are eliminated. On M68HC12 derivatives with expanded memory capability, bank numbers are specified by on-chip control registers. Since the addresses of these control registers may not be the same in all M68HC12 derivatives, the CPU12 has a dedicated control line to the on-chip integration module that indicates when a memory-expansion register is being read or written. This allows the CPU to access the PPAGE register without knowing the register address.

The indexed indirect versions of the CALL instruction access the address of the called routine and the destination page value indirectly. For other addressing mode variations of the CALL instruction, the destination page value is provided as immediate data in the instruction object code. CALL and RTC execute correctly in the normal 64-Kbyte address space, thus providing for portable code.

## **APPENDIX C**

### **HIGH-LEVEL LANGUAGE SUPPORT**

Many programmers are turning to high-level languages such as C as an alternative to coding in native assembly languages. High-level language (HLL) programming can improve productivity and produce code that is more easily maintained than assembly language programs. The most serious drawback to the use of HLL in MCUs has been the relatively large size of programs written in HLL. Larger program ROM size requirements translate into increased system costs.

Motorola solicited the cooperation of third-party software developers to assure that the CPU12 instruction set would meet the needs of a more efficient generation of compilers. Several features of the CPU12 were specifically designed to improve the efficiency of compiled HLL, and thus minimize cost.

This appendix identifies CPU12 instructions and addressing modes that provide improved support for high-level language. C language examples are provided to demonstrate how these features support efficient HLL structures and concepts. Since the CPU12 instruction set is a superset of the M68HC11 instruction set, some of the discussions use the M68HC11 as a basis for comparison.

#### **C.1 Data Types**

The CPU12 supports the bit-sized data type with bit manipulation instructions which are available in extended, direct, and indexed variations. The char data type is a simple 8-bit value that is commonly used to specify variables in a small microcontroller system because it requires less memory space than a 16-bit integer (provided the variable has a range small enough to fit into eight bits). The 16-bit CPU12 can easily handle 16-bit integer types and the available set of conditional branches (including long branches) allow branching based on signed or unsigned arithmetic results. Some of the higher math functions allow for division and multiplication involving 32-bit values, although it is somewhat less common to use such long values in a microcontroller system.

The CPU12 has special sign extension instructions to allow easy type-casting from smaller data types to larger ones, such as from char to integer. This sign extension is automatically performed when an 8-bit value is transferred to a 16-bit register.

#### **C.2 Parameters and Variables**

High-level languages make extensive use of the stack, both to pass variables and for temporary and local storage. It follows that there should be easy ways to push and pull all CPU registers, stack pointer based indexing should be allowed, and that direct arithmetic manipulation of the stack pointer value should be allowed. The CPU12 instruction set provided for all of these needs with improved indexed addressing, the addition of an LEAS instruction, and the addition of push and pull instructions for the D accumulator and the CCR.



## C.2 Parameters and Variables

High-level languages make extensive use of the stack, both to pass variables and for temporary and local storage. It follows that there should be easy ways to push and pull all CPU registers, stack pointer based indexing should be allowed, and that direct arithmetic manipulation of the stack pointer value should be allowed. The CPU12 instruction set provided for all of these needs with improved indexed addressing, the addition of an LEAS instruction, and the addition of push and pull instructions for the D accumulator and the CCR.

The CPU12 has special sign extension instructions to allow easy type-casting from smaller data types to larger ones, such as from char to integer. This sign extension is automatically performed when an 8-bit value is transferred to a 16-bit register.

The CPU12 supports the bit-sized data type with manipulation instructions which are available in extended, direct, and indexed variations. The char data type is a single 8-bit value that is commonly used to specify variables in a small microcontroller system because it requires less memory space than a 16-bit integer (provided the variable has a range small enough to fit into eight bits). The 16-bit CPU12 can easily handle the 16-bit integer types and the available set of conditional branches (including long branches) allow branching based on signed or unsigned arithmetic results. Some of the highest math functions allow for division and multiplication involving 32-bit values, although it is somewhat less common to use such long values in a microcontroller system.

## C.1 Data Types

Cussons use the M68HC11 as a basis for comparison.

CPU12 instruction set is a superset of the M68HC11 instruction set, some of the distinctive new features support efficient HLL structures and concepts. Block the This appendix identifies CPU12 instructions and addressing modes that provide improved support for high-level languages. C language examples are provided to demonstrate how these features support efficient HLL structures and concepts. Block the

cy of compiled HLL, and thus minimize cost.

Several features of the CPU12 were specifically designed to improve the efficiency of compiled HLL, and thus minimize cost.

ments translate into increased system costs.

the relatively large size of programs written in HLL. Larger program ROM size requirements translate into increased system costs.

## APPENDIX C HIGH-LEVEL LANGUAGE SUPPORT

2/15

temporarily mask off interrupts during the context switch.

The CALL instruction is similar to a JSR instruction, except that the program counter is saved on the stack and the new page value is written to the bank page register. Since the CALL instruction is unimplementable, this eliminates the need to support a destination page value as part of the instruction. When CALL executes, the old page value is saved on the stack and the new page value is written to the bank page register. The CPU13 addresses both of these problems with the unimplementable CALL and return from call (HTC) instructions.

Bank switching is a fairly common way of adapting a CPU with a 16-bit address bus to accommodate more than 64-Kbytes of program memory space. One of the most significant drawbacks of this technique has been the requirement to mask (disable) interrupts while the bank page value was being changed. Another problem is that the physical location of the bank page register can change from one MCU derivative to another (or even due to a change in mapping controls by a user program). In these situations, an operating system program has to keep track of the physical location of the page register. The CPU13 addresses both of these problems with the unimplementable CALL and return from call (HTC) instructions.

### C.3 Function Calls

The CPU13 supports pointers by allowing the use of the 16-bit indirect addressing modes (LEAX, LEAY, and LEAX, LEAY instructions) and by allowing indirect addressing modes.

### C.7 Pointers

Case and switch statements (and computed GOTOs) can use PC-relative indirect addressing to determine which path to take. Depending upon the situation, cases can use either the constant offset variation or the accumulator D offset variation of indirect addressing.

### C.8 Cases and Switch Statements

Long branches use a 16-bit relative offset that allows conditional branching to any location in the 64-Kbyte map. Short branches use a single byte relative offset that allows branching to a destination within about 4128 locations from the instruction. Short branches use a single byte relative offset that allows conditional branching to any location in the 64-Kbyte map. Long branches use a 16-bit relative offset that allows conditional branching to any location in the 64-Kbyte map.

### C.9 Conditional If Constructs

Use native instructions instead of relatively large, slow routines. Application requires these functions, the code is much more efficient if the MCU can guess because these functions can be performed as library functions. However, if an action is still instructions is not necessarily a requirement for supporting high-level action. It can be minimized by putting the results in CPU registers. Having higher-grade operands is also a potential problem in the extended multiply operations and the

Operand size is also a potential problem in the extended multiply operations but the difficulty can be minimized by putting the results in CPU registers. Having higher precision math instructions is not necessarily a requirement for supporting high-level language because these functions can be performed as library functions. However, if an application requires these functions, the code is much more efficient if the MCU can use native instructions instead of relatively large, slow routines.

### **C.5 Conditional If Constructs**

In the CPU12 instruction set, most arithmetic and data manipulation instructions automatically update the condition code register, unlike other architectures that only change condition codes during a few specific compare instructions. The CPU12 includes branch instructions that perform conditional branching based on the state of the indicators in the condition codes register. Short branches use a single byte relative offset that allows branching to a destination within about  $\pm 128$  locations from the branch. Long branches use a 16-bit relative offset that allows conditional branching to any location in the 64-Kbyte map.

### **C.6 Case and Switch Statements**

Case and switch statements (and computed GOTOs) can use PC-relative indirect addressing to determine which path to take. Depending upon the situation, cases can use either the constant offset variation or the accumulator D offset variation of indirect indexed addressing.

### **C.7 Pointers**

The CPU12 supports pointers by allowing direct arithmetic operations on the 16-bit index registers (LEAS, LEAX, and LEAY instructions) and by allowing indexed indirect addressing modes.

### **C.8 Function Calls**

Bank switching is a fairly common way of adapting a CPU with a 16-bit address bus to accommodate more than 64-Kbytes of program memory space. One of the most significant drawbacks of this technique has been the requirement to mask (disable) interrupts while the bank page value was being changed. Another problem is that the physical location of the bank page register can change from one MCU derivative to another (or even due to a change to mapping controls by a user program). In these situations, an operating system program has to keep track of the physical location of the page register. The CPU12 addresses both of these problems with the uninterruptible CALL and return from call (RTC) instructions.

The CALL instruction is similar to a JSR instruction, except that the programmer supplies a destination page value as part of the instruction. When CALL executes, the old page value is saved on the stack and the new page value is written to the bank page register. Since the CALL instruction is uninterruptible, this eliminates the need to separately mask off interrupts during the context switch.

The CPU12 has dedicated signal lines that allow the CPU to access the bank page register without having to use an address in the normal 64-Kbyte address space. This eliminates the need for the program to know where the page register is physically located.

The RTC instruction is similar to the RTS instruction, except that RTC uses the byte of information that was saved on the stack by the corresponding CALL instruction to restore the bank page register to its old value. Although a CALL/RTC pair can be used to access any function subroutine regardless of the location of the called routine (on the current bank page or a different page), it is most efficient to access some subroutines with JSR/RTS instructions when the called subroutine is on the current page or in an area of memory that is always visible in the 64-Kbyte map regardless of the bank page selection.

Push and pull instructions can be used to stack some or all the CPU registers during a function call. The CPU12 can push and pull any of the CPU registers A, B, CCR, D, X, Y, or SP.

### **C.9 Instruction Set Orthogonality**

One very helpful aspect of the CPU12 instruction set, orthogonality, is difficult to quantify in terms of direct benefit to an HLL compiler. Orthogonality refers to the regularity of the instruction set. A completely orthogonal instruction set would allow any instruction to operate in any addressing mode, would have identical code sizes and execution times for similar operations on different registers, and would include both signed and unsigned versions of all mathematical instructions. Greater regularity of the instruction makes it possible to implement compilers more efficiently, because operation is more consistent, and fewer special cases must be handled.

The CPU12 has dedicated signal lines that allow the CPU to access the bank page register without having to use an address in the normal 64-Kbyte address space. This eliminates the need for the program to know where the page register is physically located.

The RTC instruction is similar to the RTS instruction, except that RTC uses the byte of information that was saved on the stack by the corresponding CALL instruction to restore the bank page register to its old value. Although a CALL/RTC pair can be used to access any function subroutine regardless of the location of the called routine (on the current bank page or a different page), it is most efficient to access some subroutines with JSR/RTS instructions when the called routine is on the current page or in an area of memory that is always visible in the 64-Kbyte map regardless of the bank page selection.

Push and pull instructions can be used to stack some or all the CPU registers during a function call. The CPU12 can push and pull any of the CPU registers A, B, CCR, D, X, Y, or SP.

### 3.3 Instruction Set Orthogonality

One very helpful aspect of the CPU12 instruction set, orthogonality, is difficult to quantify in terms of direct benefit to an HLL compiler. Orthogonality refers to the regularity of the instruction set. A completely orthogonal instruction set would allow any instruction to operate in any addressing mode, would have identical code sizes and execution times for similar operations on different registers, and would include both signed and unsigned versions of all mathematical instructions. Greater regularity of the instruction makes it possible to implement compilers more efficiently, because operation is more consistent, and fewer special cases must be handled.

## APPENDIX D ASSEMBLY LISTING

The following assembler test file illustrates all possible variations of the M68HC12 instruction set and can be used as a quick reference for instruction syntax. Instructions are in alphabetical order and include redundancy.

*	68HC12	assembly listing	d11b a9 2f	adca	1,-x
		*	d11d a9 6f	adca	1,-y
0072	immed	equ \$72	d11f a9 a8	adca	8,-sp
0055	dir	equ \$55	d121 a9 28	adca	8,-x
1234	ext	equ \$1234	d123 a9 68	adca	8,-y
0037	ind	equ \$37	d125 a9 9f	adca	-1,sp
000e	small	equ \$e	d127 a9 1f	adca	-1,x
00cc	mask	equ \$11001100	d129 a9 5f	adca	-1,y
		*	d12b a9 90	adca	-16,sp
		*	d12d a9 10	adca	-16,x
		*	d12f a9 50	adca	-16,y
d000	ORG	\$D000	d131 a9 f1 ef	adca	-17,sp
			d134 a9 e1 ef	adca	-17,x
			d137 a9 e9 ef	adca	-17,y
			d13a a9 d2	adca	-small,pc
			d13c a9 92	adca	-small,sp
d000 00 02	dw	2	d13e a9 12	adca	-small,x
d002 02	db	2	d140 a9 52	adca	-small,y
d003 00 02	dc.w	2	d142 a9 c0	adca	0,pc
d005 02	dc.b	2	d144 a9 80	adca	0,sp
d006 02	fc b	2	d146 a9 00	adca	0,x
d007 08 ae	fdb	2222	d148 a9 40	adca	0,y
d009	ds	34	d14a a9 b0	adca	1,sp+
d02b	ds.b	34	d14c a9 30	adca	1,x+
d04d	ds.w	34	d14e a9 e2 01 88	adca	ext,x
d091	rmb	34	d152 a9 e2 89 44	adca	ext,x
d0b3	rmw	34	d156 a9 e2 33 33	adca	ext,x
			d15a a9 e2 44 44	adca	ext,x
			d15e a9 e2 01 88	adca	ext,x
			d162 a9 70	adca	1,y+
			d164 a9 81	adca	1,sp
d0f7 18 06	aba		d166 a9 01	adca	1,x
d0f9 1a e5	abx		d168 a9 41	adca	1,y
d0fb 19 ed	aby		d16a a9 bf	adca	1,sp-
d0fd 89 72	adca	#immed	d16c a9 3f	adca	1,x-
d0ff 89 72	adca	#immed	d16e a9 7f	adca	1,y-
d101 89 72	adca	#immed	d170 a9 f8 7d	adca	125,pc
d103 89 72	adca	#immed	d173 a9 f0 7d	adca	125,sp
d105 89 72	adca	#immed	d176 a9 e0 7d	adca	125,x
d107 a9 a0	adca	1,+sp	d179 a9 e8 7d	adca	125,y
d109 a9 20	adca	1,+x	d17c a9 8f	adca	15,sp
d10b a9 60	adca	1,+y	d17e a9 0f	adca	15,x
d10d a9 a7	adca	8,+sp	d180 a9 4f	adca	15,y
d10f a9 67	adca	8,+y	d182 a9 f0 10	adca	16,sp
d111 a9 c0	adca	,pc	d185 a9 e0 10	adca	16,x
d113 a9 80	adca	,sp	d188 a9 e8 10	adca	16,y
d115 a9 00	adca	,x	d18b a9 b7	adca	8,sp+
d117 a9 40	adca	,y	d18d a9 37	adca	8,x+
d119 a9 af	adca	1,-sp	d18f a9 77	adca	8,y+

d191 a9 b8	adca	8,sp-	d22c 57	asrb	
d193 a9 38	adca	8,x-	d22d 24 fe	bcc	*
d195 a9 78	adca	8,y-	d22f 25 fe	bcs	*
d197 a9 f4	adca	a,sp	d231 27 fe	beq	*
d199 a9 e4	adca	a,x	d233 2c fe	bge	*
d19b a9 ec	adca	a,y	d235 2e fe	bgt	*
d19d a9 f5	adca	b,sp	d237 22 fe	bhi	*
d19f a9 e5	adca	b,x	d239 85 72	bita	#immed
d1a1 a9 ed	adca	b,y	d23b a5 a0	bita	1,+sp
d1a3 a9 f6	adca	d,sp	d23d 95 55	bita	dir
d1a5 a9 e6	adca	d,x	d23f b5 01 88	bita	ext
d1a7 a9 ee	adca	d,y	d242 c5 72	bitb	#immed
d1a9 99 55	adca	dir	d244 e5 a0	bitb	1,+sp
d1ab 99 55	adca	dir	d246 d5 55	bitb	dir
d1ad b9 01 88	adca	ext	d248 f5 01 88	bitb	ext
d1b0 b9 01 88	adca	ext	d24b 2f fe	ble	*
d1b3 a9 f2 01 88	adca	ext,sp	d24d 23 fe	bls	*
d1b7 a9 e2 01 88	adca	ext,x	d24f 2d fe	blt	*
d1bb a9 ea 01 88	adca	ext,y	d251 2b fe	bmi	*
d1bf a9 f8 37	adca	ind,pc	d253 26 fe	bne	*
d1c2 a9 f0 37	adca	ind,sp	d255 2a fe	bpl	*
d1c5 a9 e0 37	adca	ind,x	d257 20 fe	bra	*
d1c8 a9 e8 37	adca	ind,y	d259 21 fe	brn	*
d1cb a9 ce	adca	small,pc	d25b 07 fe	bsr	*
d1cd a9 8e	adca	small,sp	d25d 28 fe	bvc	*
d1cf a9 0e	adca	small,x	d25f 29 fe	bvs	*
d1d1 a9 4e	adca	small,y	d261 0d a0 55	bclr	1,+sp \$55
d1d3 c9 72	adcb	#immed	d264 0d a0 55	bclr	1,+sp #\$55
d1d5 e9 a0	adcb	1,+sp	d267 0d a0 55	bclr	1,+sp,\$55
d1d7 e9 d2	adcb	-small,pc	d26a 0d bf 55	bclr	1,sp-,\$55
d1d9 e9 f8 7d	adcb	125,pc	d26d 0d bf 55	bclr	1,sp- #\$55
d1dc d9 55	adcb	dir	d270 0d 20 55	bclr	1,+x \$55
d1de f9 01 88	adcb	ext	d273 0d 20 55	bclr	1,+x #\$55
d1e1 e9 f2 01 88	adcb	ext,sp	d276 0d 20 55	bclr	1,+x,\$55
d1e5 8b 72	adda	#immed	d279 0d 20 55	bclr	1,+x,\$55
d1e7 ab a0	adda	1,+sp	d27c 4d 55 55	bclr	dir \$55
d1e9 9b 55	adda	dir	d27f 4d 55 55	bclr	dir #\$55
d1eb bb 01 88	adda	ext	d282 4d 55 55	bclr	dir,\$55
d1ee bb 01 88	adda	ext	d285 4d 55 55	bclr	dir,\$55
d1f1 cb 72	addb	#immed	d288 1d 01 88 55	bclr	ext \$55
d1f3 eb a0	addb	1,+sp	d28c 1d 01 88 55	bclr	ext #\$55
d1f5 db 55	addb	dir	d290 1d 01 88 55	bclr	ext,\$55
d1f7 fb 01 88	addb	ext	d294 1d 01 88 55	bclr	ext,\$55
d1fa c3 00 72	addd	#immed	d298 0f a0 55 fc	brclr	1,+sp \$55 *
d1fd e3 a0	addd	1,+sp	d29c 0f a0 55 fc	brclr	1,+sp #\$55 *
d1ff d3 55	addd	dir	d2a0 0f a0 55 fc	brclr	1,+sp,\$55 *
d201 f3 01 88	addd	ext	d2a4 0f a0 55 fc	brclr	1,+sp,\$55 *
d204 84 72	anda	#immed	d2a8 4f 55 55 fc	brclr	dir \$55 *
d206 a4 a0	anda	1,+sp	d2ac 4f 55 55 fc	brclr	dir #\$55 *
d208 94 55	anda	dir	d2b0 4f 55 55 fc	brclr	dir,\$55 *
d20a b4 01 88	anda	ext	d2b4 4f 55 55 fc	brclr	dir,\$55 *
d20d c4 72	andb	#immed	d2b8 1f 01 88 55 fb	brclr	ext \$55 *
d20f e4 a0	andb	1,+sp	d2bd 1f 01 88 55 fb	brclr	ext #\$55 *
d211 d4 55	andb	dir	d2c2 1f 01 88 55 fb	brclr	ext,\$55,*
d213 f4 01 88	andb	ext	d2c7 1f 01 88 55 fb	brclr	ext,\$55,*
d216 10 72	andcc	#immed	d2cc 0e a0 55 fc	brset	1,+sp \$55 *
d218 68 a0	asl	1,+sp	d2d0 0e a0 55 fc	brset	1,+sp #\$55 *
d21a 78 00 55	asl	dir	d2d4 0e a0 55 fc	brset	1,+sp,\$55,*
d21d 78 01 88	asl	ext	d2d8 0e a0 55 fc	brset	1,+sp,\$55,*
d220 48	asla				
d221 58	aslb				
d222 59	asld				
d223 67 a0	asr	1,+sp			
d225 77 00 55	asr	dir			
d228 77 01 88	asr	ext			
d22b 47	asra				

```

d2dc 4e 55 55 fc      brset   dir $55 *
d2e0 4e 55 55 fc      brset   dir, $55 *
d2e4 4e 55 55 fc      brset   dir, $55, *
d2e8 4e 55 55 fc      brset   dir, $55, *

d2ec 1e 01 88 55 fb   brset   ext $55 *
d2f1 1e 01 88 55 fb   brset   ext $55 *
d2f6 1e 01 88 55 fb   brset   ext, $55, *
d2fb 1e 01 88 55 fb   brset   ext, $55, *

d300 0c a0 55         bset     1, +sp $55
d303 0c a0 55         bset     1, +sp $55
d306 0c a0 55         bset     1, +sp, $55
d309 0c a0 55         bset     1, +sp, $55

d30c 4c 55 55         bset     dir $55
d30f 4c 55 55         bset     dir $55
d312 4c 55 55         bset     dir, $55
d315 4c 55 55         bset     dir, $55

d318 1c 01 88 55     bset     ext $55
d31c 1c 01 88 55     bset     ext $55
d320 1c 01 88 55     bset     ext, $55
d324 1c 01 88 55     bset     ext, $55

d328 4b a0 55        call     1, +sp $55
d32b 4b 20 55        call     1, +x $55
d32e 4b 60 55        call     1, +y $55
d331 4b a7 55        call     8, +sp $55
d334 4b 27 55        call     8, +x $55
d337 4b 67 55        call     8, +y $55
d33a 4b c0 55        call     ,pc $55
d33d 4b 80 55        call     ,sp $55
d340 4b 00 55        call     ,x $55
d343 4b 40 55        call     ,y $55
d346 4b af 55        call     1, -sp $55
d349 4b 2f 55        call     1, -x $55
d34c 4b 6f 55        call     1, -y $55
d34f 4b a8 55        call     8, -sp $55
d352 4b 28 55        call     8, -x $55
d355 4b 68 55        call     8, -y $55
d358 4b 9f 55        call     -1, sp $55
d35b 4b 1f 55        call     -1, x $55
d35e 4b 5f 55        call     -1, y $55
d361 4b 90 55        call     -16, sp $55
d364 4b 10 55        call     -16, x $55
d367 4b 50 55        call     -16, y $55
d36a 4b f1 ef 55     call     -17, sp $55
d36e 4b e1 ef 55     call     -17, x $55
d372 4b e9 ef 55     call     -17, y $55
d376 4b d2 55        call     -small, pc $55
d379 4b 92 55        call     -small, sp $55
d37c 4b 12 55        call     -small, x $55
d37f 4b 52 55        call     -small, y $55
d382 4b c0 55        call     0, pc $55
d385 4b 80 55        call     0, sp $55
d388 4b 00 55        call     0, x $55
d38b 4b 40 55        call     0, y $55
d38e 4b b0 55        call     1, sp+ $55
d391 4b 30 55        call     1, x+ $55
d394 4b 70 55        call     1, y+ $55
d397 4b 81 55        call     1, sp $55
d39a 4b 01 55        call     1, x $55
d39d 4b 41 55        call     1, y $55
d3a0 4b bf 55        call     1, sp- $55

```

```

d3a3 4b 3f 55        call     1, x- $55
d3a6 4b 7f 55        call     1, y- $55
d3a9 4b f8 7d 55     call     125, pc $55
d3ad 4b f0 7d 55     call     125, sp $55
d3b1 4b e0 7d 55     call     125, x $55
d3b5 4b e8 7d 55     call     125, y $55
d3b9 4b 8f 55        call     15, sp $55
d3bc 4b 0f 55        call     15, x $55
d3bf 4b 4f 55        call     15, y $55
d3c2 4b f0 10 55     call     16, sp $55
d3c6 4b e0 10 55     call     16, x $55
d3ca 4b e8 10 55     call     16, y $55
d3ce 4b b7 55        call     8, sp+ $55
d3d1 4b 37 55        call     8, x+ $55
d3d4 4b 77 55        call     8, y+ $55
d3d7 4b b8 55        call     8, sp- $55
d3da 4b 38 55        call     8, x- $55
d3dd 4b 78 55        call     8, y- $55
d3e0 4b f4 55        call     a, sp $55
d3e3 4b e4 55        call     a, x $55
d3e6 4b ec 55        call     a, y $55
d3e9 4b f5 55        call     b, sp $55
d3ec 4b e5 55        call     b, x $55
d3ef 4b ed 55        call     b, y $55
d3f2 4b f6 55        call     d, sp $55
d3f5 4b e6 55        call     d, x $55
d3f8 4b ee 55        call     d, y $55
d3fb 4a 00 55 55     call     dir $55
d3ff 4a 01 88 55     call     ext $55
d403 4b f2 01 88 55  call     ext, sp $55
d408 4b e2 01 88 55  call     ext, x $55
d40d 4b ea 01 88 55  call     ext, y $55
d412 4b f8 37 55     call     ind, pc $55
d416 4b f0 37 55     call     ind, sp $55
d41a 4b e0 37 55     call     ind, x $55
d41e 4b e8 37 55     call     ind, y $55
d422 4b ce 55        call     small, pc $55
d425 4b 8e 55        call     small, sp $55
d428 4b 0e 55        call     small, x $55
d42b 4b 4e 55        call     small, y $55

d42e 18 17          cba
d430 10 fe          clc
d432 10 ef          cli
d434 69 a0          clr     1, +sp
d436 79 00 55       clr     dir
d439 79 01 88       clr     ext
d43c 87             clra
d43d c7             clrb
d43e 10 fd          clv
d440 81 72          cmpa   #immed
d442 a1 a0          cmpa   1, +sp
d444 91 55          cmpa   dir
d446 b1 01 88       cmpa   ext

d449 c1 72          cmpb   #immed
d44b c1 72          cmpb   #immed
d44d e1 a0          cmpb   1, +sp
d44f e1 20          cmpb   1, +x
d451 e1 60          cmpb   1, +y
d453 e1 a7          cmpb   8, +sp
d455 e1 27          cmpb   8, +x
d457 e1 67          cmpb   8, +y
d459 e1 c0          cmpb   ,pc
d45b e1 80          cmpb   ,sp
d45d e1 00          cmpb   ,x

```



d45f e1 40	cmpb	,y	d4f6 e1 f0 37	cmpb	ind,sp
d461 e1 af	cmpb	1,-sp	d4f9 e1 e0 37	cmpb	ind,x
d463 e1 2f	cmpb	1,-x	d4fc e1 e8 37	cmpb	ind,y
d465 e1 6f	cmpb	1,-y	d4ff e1 ce	cmpb	small,pc
d467 e1 a8	cmpb	8,-sp	d501 e1 8e	cmpb	small,sp
d469 e1 28	cmpb	8,-x	d503 e1 0e	cmpb	small,x
d46b e1 68	cmpb	8,-y	d505 e1 4e	cmpb	small,y
d46d e1 9f	cmpb	-1,sp	d507 61 a0	com	1,+sp
d46f e1 1f	cmpb	-1,x	d509 61 20	com	1,+x
d471 e1 5f	cmpb	-1,y	d50b 61 60	com	1,+y
d473 e1 90	cmpb	-16,sp	d50d 61 a7	com	8,+sp
d475 e1 10	cmpb	-16,x	d50f 61 27	com	8,+x
d477 e1 50	cmpb	-16,y	d511 61 67	com	8,+y
d479 e1 f1 ef	cmpb	-17,sp	d513 61 c0	com	,pc
d47c e1 e1 ef	cmpb	-17,x	d515 61 80	com	,sp
d47f e1 e9 ef	cmpb	-17,y	d517 61 00	com	,x
d482 e1 d2	cmpb	-small,pc	d519 61 40	com	,y
d484 e1 92	cmpb	-small,sp	d51b 61 af	com	1,-sp
d486 e1 12	cmpb	-small,x	d51d 61 2f	com	1,-x
d488 e1 52	cmpb	-small,y	d51f 61 6f	com	1,-y
d48a e1 c0	cmpb	0,pc	d521 61 a8	com	8,-sp
d48c e1 80	cmpb	0,sp	d523 61 28	com	8,-x
d48e e1 00	cmpb	0,x	d525 61 68	com	8,-y
d490 e1 40	cmpb	0,y	d527 61 9f	com	-1,sp
d492 e1 b0	cmpb	1,sp+	d529 61 1f	com	-1,x
d494 e1 30	cmpb	1,x+	d52b 61 5f	com	-1,y
d496 e1 70	cmpb	1,y+	d52d 61 90	com	-16,sp
d498 e1 81	cmpb	1,sp	d52f 61 10	com	-16,x
d49a e1 01	cmpb	1,x	d531 61 50	com	-16,y
d49c e1 41	cmpb	1,y	d533 61 f1 ef	com	-17,sp
d49e e1 bf	cmpb	1,sp-	d536 61 e1 ef	com	-17,x
d4a0 e1 3f	cmpb	1,x-	d539 61 e9 ef	com	-17,y
d4a2 e1 7f	cmpb	1,y-	d53c 61 d2	com	-small,pc
d4a4 e1 f8 7d	cmpb	125,pc	d53e 61 92	com	-small,sp
d4a7 e1 f0 7d	cmpb	125,sp	d540 61 12	com	-small,x
d4aa e1 e0 7d	cmpb	125,x	d542 61 52	com	-small,y
d4ad e1 e8 7d	cmpb	125,y	d544 61 c0	com	0,pc
d4b0 e1 8f	cmpb	15,sp	d546 61 80	com	0,sp
d4b2 e1 0f	cmpb	15,x	d548 61 00	com	0,x
d4b4 e1 4f	cmpb	15,y	d54a 61 40	com	0,y
d4b6 e1 f0 10	cmpb	16,sp	d54c 61 b0	com	1,sp+
d4b9 e1 e0 10	cmpb	16,x	d54e 61 30	com	1,x+
d4bc e1 e8 10	cmpb	16,y	d550 61 70	com	1,y+
d4bf e1 b7	cmpb	8,sp+	d552 61 81	com	1,sp
d4c1 e1 37	cmpb	8,x+	d554 61 01	com	1,x
d4c3 e1 77	cmpb	8,y+	d556 61 41	com	1,y
d4c5 e1 b8	cmpb	8,sp-	d558 61 bf	com	1,sp-
d4c7 e1 38	cmpb	8,x-	d55a 61 3f	com	1,x-
d4c9 e1 78	cmpb	8,y-	d55c 61 7f	com	1,y-
d4cb e1 f4	cmpb	a,sp	d55e 61 f8 7d	com	125,pc
d4cd e1 e4	cmpb	a,x	d561 61 f0 7d	com	125,sp
d4cf e1 ec	cmpb	a,y	d564 61 e0 7d	com	125,x
d4d1 e1 f5	cmpb	b,sp	d567 61 e8 7d	com	125,y
d4d3 e1 e5	cmpb	b,x	d56a 61 8f	com	15,sp
d4d5 e1 ed	cmpb	b,y	d56c 61 0f	com	15,x
d4d7 e1 f6	cmpb	d,sp	d56e 61 4f	com	15,y
d4d9 e1 e6	cmpb	d,x	d570 61 f0 10	com	16,sp
d4db e1 ee	cmpb	d,y	d573 61 e0 10	com	16,x
d4dd d1 55	cmpb	dir	d576 61 e8 10	com	16,y
d4df d1 55	cmpb	dir	d579 61 b7	com	8,sp+
d4e1 f1 01 88	cmpb	ext	d57b 61 37	com	8,x+
d4e4 f1 01 88	cmpb	ext	d57d 61 77	com	8,y+
d4e7 e1 f2 01 88	cmpb	ext,sp	d57f 61 b8	com	8,sp-
d4eb e1 e2 01 88	cmpb	ext,x	d581 61 38	com	8,x-
d4ef e1 ea 01 88	cmpb	ext,y	d583 61 78	com	8,y-
d4f3 e1 f8 37	cmpb	ind,pc	d585 61 f4	com	a,sp

d587 61 e4	com	a,x	d61b ac 3f	cpd	1,x-
d589 61 ec	com	a,y	d61d ac 7f	cpd	1,y-
d58b 61 f5	com	b,sp	d61f ac f8 7d	cpd	125,pc
d58d 61 e5	com	b,x	d622 ac f0 7d	cpd	125,sp
d58f 61 ed	com	b,y	d625 ac e0 7d	cpd	125,x
d591 61 f6	com	d,sp	d628 ac e8 7d	cpd	125,y
d593 61 e6	com	d,x	d62b ac 8f	cpd	15,sp
d595 61 ee	com	d,y	d62d ac 0f	cpd	15,x
d597 71 00 55	com	dir	d62f ac 4f	cpd	15,y
d59a 71 01 88	com	ext	d631 ac f0 10	cpd	16,sp
d59d 71 01 88	com	ext	d634 ac e0 10	cpd	16,x
d5a0 61 f2 01 88	com	ext,sp	d637 ac e8 10	cpd	16,y
d5a4 61 e2 01 88	com	ext,x	d63a ac b7	cpd	8,sp+
d5a8 61 ea 01 88	com	ext,y	d63c ac 37	cpd	8,x+
d5ac 61 f8 37	com	ind,pc	d63e ac 77	cpd	8,y+
d5af 61 f0 37	com	ind,sp	d640 ac b8	cpd	8,sp-
d5b2 61 e0 37	com	ind,x	d642 ac 38	cpd	8,x-
d5b5 61 e8 37	com	ind,y	d644 ac 78	cpd	8,y-
d5b8 61 ce	com	small,pc	d646 ac f4	cpd	a,sp
d5ba 61 8e	com	small,sp	d648 ac e4	cpd	a,x
d5bc 61 0e	com	small,x	d64a ac ec	cpd	a,y
d5be 61 4e	com	small,y	d64c ac f5	cpd	b,sp
d5c0 41	coma		d64e ac e5	cpd	b,x
d5c1 51	comb		d650 ac ed	cpd	b,y
d5c2 8c 00 72	cpd	#immed	d652 ac f6	cpd	d,sp
d5c5 8c 00 72	cpd	#immed	d654 ac e6	cpd	d,x
d5c8 ac a0	cpd	1,+sp	d656 ac ee	cpd	d,y
d5ca ac 20	cpd	1,+x	d658 9c 55	cpd	dir
d5cc ac 60	cpd	1,+y	d65a 9c 55	cpd	dir
d5ce ac a7	cpd	8,+sp	d65c bc 01 88	cpd	ext
d5d0 ac 27	cpd	8,+x	d65f bc 01 88	cpd	ext
d5d2 ac 67	cpd	8,+y	d662 ac f2 01 88	cpd	ext,sp
d5d4 ac c0	cpd	,pc	d666 ac e2 01 88	cpd	ext,x
d5d6 ac 80	cpd	,sp	d66a ac ea 01 88	cpd	ext,y
d5d8 ac 00	cpd	,x	d66e ac f8 37	cpd	ind,pc
d5da ac 40	cpd	,y	d671 ac f0 37	cpd	ind,sp
d5dc ac af	cpd	1,-sp	d674 ac e0 37	cpd	ind,x
d5de ac 2f	cpd	1,-x	d677 ac e8 37	cpd	ind,y
d5e0 ac 6f	cpd	1,-y	d67a ac ce	cpd	small,pc
d5e2 ac a8	cpd	8,-sp	d67c ac 8e	cpd	small,sp
d5e4 ac 28	cpd	8,-x	d67e ac 0e	cpd	small,x
d5e6 ac 68	cpd	8,-y	d680 ac 4e	cpd	small,y
d5e8 ac 9f	cpd	-1,sp	d682 8f 00 72	cps	#immed
d5ea ac 1f	cpd	-1,x	d685 af a0	cps	1,+sp
d5ec ac 5f	cpd	-1,y	d687 af 20	cps	1,+x
d5ee ac 90	cpd	-16,sp	d689 af 60	cps	1,+y
d5f0 ac 10	cpd	-16,x	d68b af a7	cps	8,+sp
d5f2 ac 50	cpd	-16,y	d68d af 27	cps	8,+x
d5f4 ac f1 ef	cpd	-17,sp	d68f af 67	cps	8,+y
d5f7 ac e1 ef	cpd	-17,x	d691 af c0	cps	,pc
d5fa ac e9 ef	cpd	-17,y	d693 af 80	cps	,sp
d5fd ac d2	cpd	-small,pc	d695 af 00	cps	,x
d5ff ac 92	cpd	-small,sp	d697 af 40	cps	,y
d601 ac 12	cpd	-small,x	d699 af af	cps	1,-sp
d603 ac 52	cpd	-small,y	d69b af 2f	cps	1,-x
d605 ac c0	cpd	0,pc	d69d af 6f	cps	1,-y
d607 ac 80	cpd	0,sp	d69f af a8	cps	8,-sp
d609 ac 00	cpd	0,x	d6a1 af 28	cps	8,-x
d60b ac 40	cpd	0,y	d6a3 af 68	cps	8,-y
d60d ac b0	cpd	1,sp+	d6a5 af 9f	cps	-1,sp
d60f ac 30	cpd	1,x+	d6a7 af 1f	cps	-1,x
d611 ac 70	cpd	1,y+	d6a9 af 5f	cps	-1,y
d613 ac 81	cpd	1,sp	d6ab af 90	cps	-16,sp
d615 ac 01	cpd	1,x	d6ad af 10	cps	-16,x
d617 ac 41	cpd	1,y	d6af af 50	cps	-16,y
d619 ac bf	cpd	1,sp-	d6b1 af f1 ef	cps	-17,sp

d6b4 af e1 ef	cps	-17,x	d74f ae 67	cpx	8,+y
d6b7 af e9 ef	cps	-17,y	d751 ae c0	cpx	,pc
d6ba af d2	cps	-small,pc	d753 ae 80	cpx	,sp
d6bc af 92	cps	-small,sp	d755 ae 00	cpx	,x
d6be af 12	cps	-small,x	d757 ae 40	cpx	,y
d6c0 af 52	cps	-small,y	d759 ae af	cpx	1,-sp
d6c2 af c0	cps	0,pc	d75b ae 2f	cpx	1,-x
d6c4 af 80	cps	0,sp	d75d ae 6f	cpx	1,-y
d6c6 af 00	cps	0,x	d75f ae a8	cpx	8,-sp
d6c8 af 40	cps	0,y	d761 ae 28	cpx	8,-x
d6ca af b0	cps	1,sp+	d763 ae 68	cpx	8,-y
d6cc af 30	cps	1,x+	d765 ae 9f	cpx	-1,sp
d6ce af 70	cps	1,y+	d767 ae 1f	cpx	-1,x
d6d0 af 81	cps	1,sp	d769 ae 5f	cpx	-1,y
d6d2 af 01	cps	1,x	d76b ae 90	cpx	-16,sp
d6d4 af 41	cps	1,y	d76d ae 10	cpx	-16,x
d6d6 af bf	cps	1,sp-	d76f ae 50	cpx	-16,y
d6d8 af 3f	cps	1,x-	d771 ae f1 ef	cpx	-17,sp
d6da af 7f	cps	1,y-	d774 ae e1 ef	cpx	-17,x
d6dc af f8 7d	cps	125,pc	d777 ae e9 ef	cpx	-17,y
d6df af f0 7d	cps	125,sp	d77a ae d2	cpx	-small,pc
d6e2 af e0 7d	cps	125,x	d77c ae 92	cpx	-small,sp
d6e5 af e8 7d	cps	125,y	d77e ae 12	cpx	-small,x
d6e8 af 8f	cps	15,sp	d780 ae 52	cpx	-small,y
d6ea af 0f	cps	15,x	d782 ae c0	cpx	0,pc
d6ec af 4f	cps	15,y	d784 ae 80	cpx	0,sp
d6ee af f0 10	cps	16,sp	d786 ae 00	cpx	0,x
d6f1 af e0 10	cps	16,x	d788 ae 40	cpx	0,y
d6f4 af e8 10	cps	16,y	d78a ae b0	cpx	1,sp+
d6f7 af b7	cps	8,sp+	d78c ae 30	cpx	1,x+
d6f9 af 37	cps	8,x+	d78e ae 70	cpx	1,y+
d6fb af 77	cps	8,y+	d790 ae 81	cpx	1,sp
d6fd af b8	cps	8,sp-	d792 ae 01	cpx	1,x
d6ff af 38	cps	8,x-	d794 ae 41	cpx	1,y
d701 af 78	cps	8,y-	d796 ae bf	cpx	1,sp-
d703 af f4	cps	a,sp	d798 ae 3f	cpx	1,x-
d705 af e4	cps	a,x	d79a ae 7f	cpx	1,y-
d707 af ec	cps	a,y	d79c ae f8 7d	cpx	125,pc
d709 af f5	cps	b,sp	d79f ae f0 7d	cpx	125,sp
d70b af e5	cps	b,x	d7a2 ae e0 7d	cpx	125,x
d70d af ed	cps	b,y	d7a5 ae e8 7d	cpx	125,y
d70f af f6	cps	d,sp	d7a8 ae 8f	cpx	15,sp
d711 af e6	cps	d,x	d7aa ae 0f	cpx	15,x
d713 af ee	cps	d,y	d7ac ae 4f	cpx	15,y
d715 9f 55	cps	dir	d7ae ae f0 10	cpx	16,sp
d717 9f 55	cps	dir	d7b1 ae e0 10	cpx	16,x
d719 bf 01 88	cps	ext	d7b4 ae e8 10	cpx	16,y
d71c bf 01 88	cps	ext	d7b7 ae b7	cpx	8,sp+
d71f af f2 01 88	cps	ext,sp	d7b9 ae 37	cpx	8,x+
d723 af e2 01 88	cps	ext,x	d7bb ae 77	cpx	8,y+
d727 af ea 01 88	cps	ext,y	d7bd ae b8	cpx	8,sp-
d72b af f8 37	cps	ind,pc	d7bf ae 38	cpx	8,x-
d72e af f0 37	cps	ind,sp	d7c1 ae 78	cpx	8,y-
d731 af e0 37	cps	ind,x	d7c3 ae f4	cpx	a,sp
d734 af e8 37	cps	ind,y	d7c5 ae e4	cpx	a,x
d737 af ce	cps	small,pc	d7c7 ae ec	cpx	a,y
d739 af 8e	cps	small,sp	d7c9 ae f5	cpx	b,sp
d73b af 0e	cps	small,x	d7cb ae e5	cpx	b,x
d73d af 4e	cps	small,y	d7cd ae ed	cpx	b,y
d73f 8e 00 72	cpx	#immed	d7cf ae f6	cpx	d,sp
d742 8e 00 72	cpx	#immed	d7d1 ae e6	cpx	d,x
d745 ae a0	cpx	1,+sp	d7d3 ae ee	cpx	d,y
d747 ae 20	cpx	1,+x	d7d5 9e 55	cpx	dir
d749 ae 60	cpx	1,+y	d7d7 9e 55	cpx	dir
d74b ae a7	cpx	8,+sp	d7d9 be 01 88	cpx	ext
d74d ae 27	cpx	8,+x	d7dc be 01 88	cpx	ext

d7df	ae f2 01 88	cpx	ext,sp	d879	ad 37	cpy	8,x+
d7e3	ae e2 01 88	cpx	ext,x	d87b	ad 77	cpy	8,y+
d7e7	ae ea 01 88	cpx	ext,y	d87d	ad b8	cpy	8,sp-
d7eb	ae f8 37	cpx	ind,pc	d87f	ad 38	cpy	8,x-
d7ee	ae f0 37	cpx	ind,sp	d881	ad 78	cpy	8,y-
d7f1	ae e0 37	cpx	ind,x	d883	ad f4	cpy	a,sp
d7f4	ae e8 37	cpx	ind,y	d885	ad e4	cpy	a,x
d7f7	ae ce	cpx	small,pc	d887	ad ec	cpy	a,y
d7f9	ae 8e	cpx	small,sp	d889	ad f5	cpy	b,sp
d7fb	ae 0e	cpx	small,x	d88b	ad e5	cpy	b,x
d7fd	ae 4e	cpx	small,y	d88d	ad ed	cpy	b,y
d7ff	8d 00 72	cpy	#immed	d88f	ad f6	cpy	d,sp
d802	8d 00 72	cpy	#immed	d891	ad e6	cpy	d,x
d805	ad a0	cpy	1,+sp	d893	ad ee	cpy	d,y
d807	ad 20	cpy	1,+x	d895	9d 55	cpy	dir
d809	ad 60	cpy	1,+y	d897	9d 55	cpy	dir
d80b	ad a7	cpy	8,+sp	d899	bd 01 88	cpy	ext
d80d	ad 27	cpy	8,+x	d89c	bd 01 88	cpy	ext
d80f	ad 67	cpy	8,+y	d89f	ad f2 01 88	cpy	ext,sp
d811	ad c0	cpy	,pc	d8a3	ad e2 01 88	cpy	ext,x
d813	ad 80	cpy	,sp	d8a7	ad ea 01 88	cpy	ext,y
d815	ad 00	cpy	,x	d8ab	ad f8 37	cpy	ind,pc
d817	ad 40	cpy	,y	d8ae	ad f0 37	cpy	ind,sp
d819	ad af	cpy	1,-sp	d8b1	ad e0 37	cpy	ind,x
d81b	ad 2f	cpy	1,-x	d8b4	ad e8 37	cpy	ind,y
d81d	ad 6f	cpy	1,-y	d8b7	ad ce	cpy	small,pc
d81f	ad a8	cpy	8,-sp	d8b9	ad 8e	cpy	small,sp
d821	ad 28	cpy	8,-x	d8bb	ad 0e	cpy	small,x
d823	ad 68	cpy	8,-y	d8bd	ad 4e	cpy	small,y
d825	ad 9f	cpy	-1,sp	d8bf	18 07	daa	
d827	ad 1f	cpy	-1,x	d8c1	04 30 fd	dbne	a *
d829	ad 5f	cpy	-1,y	d8c4	04 31 fd	dbne	b *
d82b	ad 90	cpy	-16,sp	d8c7	04 35 fd	dbne	x *
d82d	ad 10	cpy	-16,x	d8ca	04 36 fd	dbne	y *
d82f	ad 50	cpy	-16,y	d8cd	63 a0	dec	1,+sp
d831	ad f1 ef	cpy	-17,sp	d8cf	63 20	dec	1,+x
d834	ad e1 ef	cpy	-17,x	d8d1	63 60	dec	1,+y
d837	ad e9 ef	cpy	-17,y	d8d3	63 a7	dec	8,+sp
d83a	ad d2	cpy	-small,pc	d8d5	63 27	dec	8,+x
d83c	ad 92	cpy	-small,sp	d8d7	63 67	dec	8,+y
d83e	ad 12	cpy	-small,x	d8d9	63 c0	dec	,pc
d840	ad 52	cpy	-small,y	d8db	63 80	dec	,sp
d842	ad c0	cpy	0,pc	d8dd	63 00	dec	,x
d844	ad 80	cpy	0,sp	d8df	63 40	dec	,y
d846	ad 00	cpy	0,x	d8e1	63 af	dec	1,-sp
d848	ad 40	cpy	0,y	d8e3	63 2f	dec	1,-x
d84a	ad b0	cpy	1,sp+	d8e5	63 6f	dec	1,-y
d84c	ad 30	cpy	1,x+	d8e7	63 a8	dec	8,-sp
d84e	ad 70	cpy	1,y+	d8e9	63 28	dec	8,-x
d850	ad 81	cpy	1,sp	d8eb	63 68	dec	8,-y
d852	ad 01	cpy	1,x	d8ed	63 9f	dec	-1,sp
d854	ad 41	cpy	1,y	d8ef	63 1f	dec	-1,x
d856	ad bf	cpy	1,sp-	d8f1	63 5f	dec	-1,y
d858	ad 3f	cpy	1,x-	d8f3	63 90	dec	-16,sp
d85a	ad 7f	cpy	1,y-	d8f5	63 10	dec	-16,x
d85c	ad f8 7d	cpy	125,pc	d8f7	63 50	dec	-16,y
d85f	ad f0 7d	cpy	125,sp	d8f9	63 f1 ef	dec	-17,sp
d862	ad e0 7d	cpy	125,x	d8fc	63 e1 ef	dec	-17,x
d865	ad e8 7d	cpy	125,y	d8ff	63 e9 ef	dec	-17,y
d868	ad 8f	cpy	15,sp	d902	63 d2	dec	-small,pc
d86a	ad 0f	cpy	15,x	d904	63 92	dec	-small,sp
d86c	ad 4f	cpy	15,y	d906	63 12	dec	-small,x
d86e	ad f0 10	cpy	16,sp	d908	63 52	dec	-small,y
d871	ad e0 10	cpy	16,x	d90a	63 c0	dec	0,pc
d874	ad e8 10	cpy	16,y	d90c	63 80	dec	0,sp
d877	ad b7	cpy	8,sp+	d90e	63 00	dec	0,x

d910 63 40	dec	0,y	d9b0 18 1a 80	emaxd	,sp
d912 63 b0	dec	1,sp+	d9b3 18 1a 00	emaxd	,x
d914 63 30	dec	1,x+	d9b6 18 1a 40	emaxd	,y
d916 63 70	dec	1,y+	d9b9 18 1a af	emaxd	1,-sp
d918 63 81	dec	1,sp	d9bc 18 1a 2f	emaxd	1,-x
d91a 63 01	dec	1,x	d9bf 18 1a 6f	emaxd	1,-y
d91c 63 41	dec	1,y	d9c2 18 1a a8	emaxd	8,-sp
d91e 63 bf	dec	1,sp-	d9c5 18 1a 28	emaxd	8,-x
d920 63 3f	dec	1,x-	d9c8 18 1a 68	emaxd	8,-y
d922 63 7f	dec	1,y-	d9cb 18 1a 9f	emaxd	-1,sp
d924 63 f8 7d	dec	125,pc	d9ce 18 1a 1f	emaxd	-1,x
d927 63 f0 7d	dec	125,sp	d9d1 18 1a 5f	emaxd	-1,y
d92a 63 e0 7d	dec	125,x	d9d4 18 1a 90	emaxd	-16,sp
d92d 63 e8 7d	dec	125,y	d9d7 18 1a 10	emaxd	-16,x
d930 63 8f	dec	15,sp	d9da 18 1a 50	emaxd	-16,y
d932 63 0f	dec	15,x	d9dd 18 1a f1 ef	emaxd	-17,sp
d934 63 4f	dec	15,y	d9e1 18 1a e1 ef	emaxd	-17,x
d936 63 f0 10	dec	16,sp	d9e5 18 1a e9 ef	emaxd	-17,y
d939 63 e0 10	dec	16,x	d9e9 18 1a d2	emaxd	-small,pc
d93c 63 e8 10	dec	16,y	d9ec 18 1a 92	emaxd	-small,sp
d93f 63 b7	dec	8,sp+	d9ef 18 1a 12	emaxd	-small,x
d941 63 37	dec	8,x+	d9f2 18 1a 52	emaxd	-small,y
d943 63 77	dec	8,y+	d9f5 18 1a c0	emaxd	0,pc
d945 63 b8	dec	8,sp-	d9f8 18 1a 80	emaxd	0,sp
d947 63 38	dec	8,x-	d9fb 18 1a 00	emaxd	0,x
d949 63 78	dec	8,y-	d9fe 18 1a 40	emaxd	0,y
d94b 63 f4	dec	a,sp	da01 18 1a b0	emaxd	1,sp+
d94d 63 e4	dec	a,x	da04 18 1a 30	emaxd	1,x+
d94f 63 ec	dec	a,y	da07 18 1a 70	emaxd	1,y+
d951 63 f5	dec	b,sp	da0a 18 1a 81	emaxd	1,sp
d953 63 e5	dec	b,x	da0d 18 1a 01	emaxd	1,x
d955 63 ed	dec	b,y	da10 18 1a 41	emaxd	1,y
d957 63 f6	dec	d,sp	da13 18 1a bf	emaxd	1,sp-
d959 63 e6	dec	d,x	da16 18 1a 3f	emaxd	1,x-
d95b 63 ee	dec	d,y	da19 18 1a 7f	emaxd	1,y-
d95d 73 00 55	dec	dir	da1c 18 1a f8 7d	emaxd	125,pc
d960 73 01 88	dec	ext	da20 18 1a f0 7d	emaxd	125,sp
d963 73 01 88	dec	ext	da24 18 1a e0 7d	emaxd	125,x
d966 63 f2 01 88	dec	ext,sp	da28 18 1a e8 7d	emaxd	125,y
d96a 63 e2 01 88	dec	ext,x	da2c 18 1a 8f	emaxd	15,sp
d96e 63 ea 01 88	dec	ext,y	da2f 18 1a 0f	emaxd	15,x
d972 63 f8 37	dec	ind,pc	da32 18 1a 4f	emaxd	15,y
d975 63 f0 37	dec	ind,sp	da35 18 1a f0 10	emaxd	16,sp
d978 63 e0 37	dec	ind,x	da39 18 1a e0 10	emaxd	16,x
d97b 63 e8 37	dec	ind,y	da3d 18 1a e8 10	emaxd	16,y
d97e 63 ce	dec	small,pc	da41 18 1a b7	emaxd	8,sp+
d980 63 8e	dec	small,sp	da44 18 1a 37	emaxd	8,x+
d982 63 0e	dec	small,x	da47 18 1a 77	emaxd	8,y+
d984 63 4e	dec	small,y	da4a 18 1a b8	emaxd	8,sp-
d986 43	deca		da4d 18 1a 38	emaxd	8,x-
d987 53	decb		da50 18 1a 78	emaxd	8,y-
d988 1b 9f	des		da53 18 1a f4	emaxd	a,sp
d98a 09	dex		da56 18 1a e4	emaxd	a,x
d98b 03	dey		da59 18 1a ec	emaxd	a,y
d98c 11	ediv		da5c 18 1a f5	emaxd	b,sp
d98d 18 14	edivs		da5f 18 1a e5	emaxd	b,x
d98f 18 12 00 55	emacs	dir	da62 18 1a ed	emaxd	b,y
d993 18 12 01 88	emacs	ext	da65 18 1a f6	emaxd	d,sp
d997 18 12 00 0e	emacs	small	da68 18 1a e6	emaxd	d,x
d99b 18 1a a0	emaxd	1,+sp	da6b 18 1a ee	emaxd	d,y
d99e 18 1a 20	emaxd	1,+x	da6e 18 1a f2 01 88	emaxd	ext,sp
d9a1 18 1a 60	emaxd	1,+y	da73 18 1a e2 01 88	emaxd	ext,x
d9a4 18 1a a7	emaxd	8,+sp	da78 18 1a ea 01 88	emaxd	ext,y
d9a7 18 1a 27	emaxd	8,+x	da7d 18 1a f8 37	emaxd	ind,pc
d9aa 18 1a 67	emaxd	8,+y	da81 18 1a f0 37	emaxd	ind,sp
d9ad 18 1a c0	emaxd	,pc	da85 18 1a e0 37	emaxd	ind,x

da89	18 1a e8 37	emaxd	ind,y	db5a	18 1e f5	emaxm	b,sp
da8d	18 1a ce	emaxd	small,pc	db5d	18 1e e5	emaxm	b,x
da90	18 1a 8e	emaxd	small,sp	db60	18 1e ed	emaxm	b,y
da93	18 1a 0e	emaxd	small,x	db63	18 1e f6	emaxm	d,sp
da96	18 1a 4e	emaxd	small,y	db66	18 1e e6	emaxm	d,x
da99	18 1e a0	emaxm	1,+sp	db69	18 1e ee	emaxm	d,y
da9c	18 1e 20	emaxm	1,+x	db6c	18 1e f2 01 88	emaxm	ext,sp
da9f	18 1e 60	emaxm	1,+y	db71	18 1e e2 01 88	emaxm	ext,x
daa2	18 1e a7	emaxm	8,+sp	db76	18 1e ea 01 88	emaxm	ext,y
daa5	18 1e 27	emaxm	8,+x	db7b	18 1e f8 37	emaxm	ind,pc
daa8	18 1e 67	emaxm	8,+y	db7f	18 1e f0 37	emaxm	ind,sp
daab	18 1e c0	emaxm	,pc	db83	18 1e e0 37	emaxm	ind,x
daae	18 1e 80	emaxm	,sp	db87	18 1e e8 37	emaxm	ind,y
dab1	18 1e 00	emaxm	,x	db8b	18 1e ce	emaxm	small,pc
dab4	18 1e 40	emaxm	,y	db8e	18 1e 8e	emaxm	small,sp
dab7	18 1e af	emaxm	1,-sp	db91	18 1e 0e	emaxm	small,x
daba	18 1e 2f	emaxm	1,-x	db94	18 1e 4e	emaxm	small,y
dabd	18 1e 6f	emaxm	1,-y	db97	18 1b a0	emind	1,+sp
dac0	18 1e a8	emaxm	8,-sp	db9a	18 1b 20	emind	1,+x
dac3	18 1e 28	emaxm	8,-x	db9d	18 1b 60	emind	1,+y
dac6	18 1e 68	emaxm	8,-y	dba0	18 1b a7	emind	8,+sp
dac9	18 1e 9f	emaxm	-1,sp	dba3	18 1b 27	emind	8,+x
dacc	18 1e 1f	emaxm	-1,x	dba6	18 1b 67	emind	8,+y
dacf	18 1e 5f	emaxm	-1,y	dba9	18 1b c0	emind	,pc
dad2	18 1e 90	emaxm	-16,sp	dbac	18 1b 80	emind	,sp
dad5	18 1e 10	emaxm	-16,x	dbaf	18 1b 00	emind	,x
dad8	18 1e 50	emaxm	-16,y	dbb2	18 1b 40	emind	,y
dadb	18 1e f1 ef	emaxm	-17,sp	dbb5	18 1b af	emind	1,-sp
dadf	18 1e e1 ef	emaxm	-17,x	dbb8	18 1b 2f	emind	1,-x
dae3	18 1e e9 ef	emaxm	-17,y	dbbb	18 1b 6f	emind	1,-y
dae7	18 1e d2	emaxm	-small,pc	dbbe	18 1b a8	emind	8,-sp
daea	18 1e 92	emaxm	-small,sp	dbc1	18 1b 28	emind	8,-x
daed	18 1e 12	emaxm	-small,x	dbc4	18 1b 68	emind	8,-y
daf0	18 1e 52	emaxm	-small,y	dbc7	18 1b 9f	emind	-1,sp
daf3	18 1e c0	emaxm	0,pc	dbca	18 1b 1f	emind	-1,x
daf6	18 1e 80	emaxm	0,sp	dbcd	18 1b 5f	emind	-1,y
daf9	18 1e 00	emaxm	0,x	dbd0	18 1b 90	emind	-16,sp
dafc	18 1e 40	emaxm	0,y	dbd3	18 1b 10	emind	-16,x
daff	18 1e b0	emaxm	1,sp+	dbd6	18 1b 50	emind	-16,y
db02	18 1e 30	emaxm	1,x+	dbd9	18 1b f1 ef	emind	-17,sp
db05	18 1e 70	emaxm	1,y+	dbdd	18 1b e1 ef	emind	-17,x
db08	18 1e 81	emaxm	1,sp	dbe1	18 1b e9 ef	emind	-17,y
db0b	18 1e 01	emaxm	1,x	dbe5	18 1b d2	emind	-small,pc
db0e	18 1e 41	emaxm	1,y	dbe8	18 1b 92	emind	-small,sp
db11	18 1e bf	emaxm	1,sp-	dbeb	18 1b 12	emind	-small,x
db14	18 1e 3f	emaxm	1,x-	dbee	18 1b 52	emind	-small,y
db17	18 1e 7f	emaxm	1,y-	dbf1	18 1b c0	emind	0,pc
db1a	18 1e f8 7d	emaxm	125,pc	dbf4	18 1b 80	emind	0,sp
db1e	18 1e f0 7d	emaxm	125,sp	dbf7	18 1b 00	emind	0,x
db22	18 1e e0 7d	emaxm	125,x	dbfa	18 1b 40	emind	0,y
db26	18 1e e8 7d	emaxm	125,y	dbfd	18 1b b0	emind	1,sp+
db2a	18 1e 8f	emaxm	15,sp	dc00	18 1b 30	emind	1,x+
db2d	18 1e 0f	emaxm	15,x	dc03	18 1b 70	emind	1,y+
db30	18 1e 4f	emaxm	15,y	dc06	18 1b 81	emind	1,sp
db33	18 1e f0 10	emaxm	16,sp	dc09	18 1b 01	emind	1,x
db37	18 1e e0 10	emaxm	16,x	dc0c	18 1b 41	emind	1,y
db3b	18 1e e8 10	emaxm	16,y	dc0f	18 1b bf	emind	1,sp-
db3f	18 1e b7	emaxm	8,sp+	dc12	18 1b 3f	emind	1,x-
db42	18 1e 37	emaxm	8,x+	dc15	18 1b 7f	emind	1,y-
db45	18 1e 77	emaxm	8,y+	dc18	18 1b f8 7d	emind	125,pc
db48	18 1e b8	emaxm	8,sp-	dc1c	18 1b f0 7d	emind	125,sp
db4b	18 1e 38	emaxm	8,x-	dc20	18 1b e0 7d	emind	125,x
db4e	18 1e 78	emaxm	8,y-	dc24	18 1b e8 7d	emind	125,y
db51	18 1e f4	emaxm	a,sp	dc28	18 1b 8f	emind	15,sp
db54	18 1e e4	emaxm	a,x	dc2b	18 1b 0f	emind	15,x
db57	18 1e ec	emaxm	a,y	dc2e	18 1b 4f	emind	15,y



dc31	18	1b	f0	10	emind	16,sp	dd07	18	1f	01	eminm	1,x
dc35	18	1b	e0	10	emind	16,x	dd0a	18	1f	41	eminm	1,y
dc39	18	1b	e8	10	emind	16,y	dd0d	18	1f	bf	eminm	1,sp-
dc3d	18	1b	b7		emind	8,sp+	dd10	18	1f	3f	eminm	1,x-
dc40	18	1b	37		emind	8,x+	dd13	18	1f	7f	eminm	1,y-
dc43	18	1b	77		emind	8,y+	dd16	18	1f	f8 7d	eminm	125,pc
dc46	18	1b	b8		emind	8,sp-	dd1a	18	1f	f0 7d	eminm	125,sp
dc49	18	1b	38		emind	8,x-	dd1e	18	1f	e0 7d	eminm	125,x
dc4c	18	1b	78		emind	8,y-	dd22	18	1f	e8 7d	eminm	125,y
dc4f	18	1b	f4		emind	a,sp	dd26	18	1f	8f	eminm	15,sp
dc52	18	1b	e4		emind	a,x	dd29	18	1f	0f	eminm	15,x
dc55	18	1b	ec		emind	a,y	dd2c	18	1f	4f	eminm	15,y
dc58	18	1b	f5		emind	b,sp	dd2f	18	1f	f0 10	eminm	16,sp
dc5b	18	1b	e5		emind	b,x	dd33	18	1f	e0 10	eminm	16,x
dc5e	18	1b	ed		emind	b,y	dd37	18	1f	e8 10	eminm	16,y
dc61	18	1b	f6		emind	d,sp	dd3b	18	1f	b7	eminm	8,sp+
dc64	18	1b	e6		emind	d,x	dd3e	18	1f	37	eminm	8,x+
dc67	18	1b	ee		emind	d,y	dd41	18	1f	77	eminm	8,y+
dc6a	18	1b	f2	01 88	emind	ext,sp	dd44	18	1f	b8	eminm	8,sp-
dc6f	18	1b	e2	01 88	emind	ext,x	dd47	18	1f	38	eminm	8,x-
dc74	18	1b	ea	01 88	emind	ext,y	dd4a	18	1f	78	eminm	8,y-
dc79	18	1b	f8	37	emind	ind,pc	dd4d	18	1f	f4	eminm	a,sp
dc7d	18	1b	f0	37	emind	ind,sp	dd50	18	1f	e4	eminm	a,x
dc81	18	1b	e0	37	emind	ind,x	dd53	18	1f	ec	eminm	a,y
dc85	18	1b	e8	37	emind	ind,y	dd56	18	1f	f5	eminm	b,sp
dc89	18	1b	ce		emind	small,pc	dd59	18	1f	e5	eminm	b,x
dc8c	18	1b	8e		emind	small,sp	dd5c	18	1f	ed	eminm	b,y
dc8f	18	1b	0e		emind	small,x	dd5f	18	1f	f6	eminm	d,sp
dc92	18	1b	4e		emind	small,y	dd62	18	1f	e6	eminm	d,x
dc95	18	1f	a0		eminm	1,+sp	dd65	18	1f	ee	eminm	d,y
dc98	18	1f	20		eminm	1,+x	dd68	18	1f	f2 01 88	eminm	ext,sp
dc9b	18	1f	60		eminm	1,+y	dd6d	18	1f	e2 01 88	eminm	ext,x
dc9e	18	1f	a7		eminm	8,+sp	dd72	18	1f	ea 01 88	eminm	ext,y
dca1	18	1f	27		eminm	8,+x	dd77	18	1f	f8 37	eminm	ind,pc
dca4	18	1f	67		eminm	8,+y	dd7b	18	1f	f0 37	eminm	ind,sp
dca7	18	1f	c0		eminm	,pc	dd7f	18	1f	e0 37	eminm	ind,x
dcaa	18	1f	80		eminm	,sp	dd83	18	1f	e8 37	eminm	ind,y
dcad	18	1f	00		eminm	,x	dd87	18	1f	ce	eminm	small,pc
dcb0	18	1f	40		eminm	,y	dd8a	18	1f	8e	eminm	small,sp
dcb3	18	1f	af		eminm	1,-sp	dd8d	18	1f	0e	eminm	small,x
dcb6	18	1f	2f		eminm	1,-x	dd90	18	1f	4e	eminm	small,y
dcb9	18	1f	6f		eminm	1,-y	dd93	88	72		eora	#immed
dcbc	18	1f	a8		eminm	8,-sp	dd95	88	72		eora	#immed
dcbf	18	1f	28		eminm	8,-x	dd97	a8	a0		eora	1,+sp
dcc2	18	1f	68		eminm	8,-y	dd99	a8	20		eora	1,+x
dcc5	18	1f	9f		eminm	-1,sp	dd9b	a8	60		eora	1,+y
dcc8	18	1f	1f		eminm	-1,x	dd9d	a8	a7		eora	8,+sp
dccb	18	1f	5f		eminm	-1,y	dd9f	a8	27		eora	8,+x
dccf	18	1f	90		eminm	-16,sp	dda1	a8	67		eora	8,+y
dcd1	18	1f	10		eminm	-16,x	dda3	a8	c0		eora	,pc
dcd4	18	1f	50		eminm	-16,y	dda5	a8	80		eora	,sp
dcd7	18	1f	f1 ef		eminm	-17,sp	dda7	a8	00		eora	,x
dcd8	18	1f	e1 ef		eminm	-17,x	dda9	a8	40		eora	,y
dcd9	18	1f	e9 ef		eminm	-17,y	ddab	a8	af		eora	1,-sp
dce3	18	1f	d2		eminm	-small,pc	ddad	a8	2f		eora	1,-x
dce6	18	1f	92		eminm	-small,sp	ddaf	a8	6f		eora	1,-y
dce9	18	1f	12		eminm	-small,x	ddb1	a8	a8		eora	8,-sp
dcec	18	1f	52		eminm	-small,y	ddb3	a8	28		eora	8,-x
dcef	18	1f	c0		eminm	0,pc	ddb5	a8	68		eora	8,-y
dcf2	18	1f	80		eminm	0,sp	ddb7	a8	9f		eora	-1,sp
dcf5	18	1f	00		eminm	0,x	ddb9	a8	1f		eora	-1,x
dcf8	18	1f	40		eminm	0,y	ddbb	a8	5f		eora	-1,y
dcfb	18	1f	b0		eminm	1,sp+	dbbd	a8	90		eora	-16,sp
dffe	18	1f	30		eminm	1,x+	dbbf	a8	10		eora	-16,x
dd01	18	1f	70		eminm	1,y+	ddc1	a8	50		eora	-16,y
dd04	18	1f	81		eminm	1,sp	ddc3	a8	f1 ef		eora	-17,sp

ddc6 a8 e1 ef	eora	-17,x	de5f e8 67	eorb	8,+y
ddc9 a8 e9 ef	eora	-17,y	de61 e8 c0	eorb	,pc
ddcc a8 d2	eora	-small,pc	de63 e8 80	eorb	,sp
ddce a8 92	eora	-small,sp	de65 e8 00	eorb	,x
ddd0 a8 12	eora	-small,x	de67 e8 40	eorb	,y
ddd2 a8 52	eora	-small,y	de69 e8 af	eorb	1,-sp
ddd4 a8 c0	eora	0,pc	de6b e8 2f	eorb	1,-x
ddd6 a8 80	eora	0,sp	de6d e8 6f	eorb	1,-y
ddd8 a8 00	eora	0,x	de6f e8 a8	eorb	8,-sp
ddda a8 40	eora	0,y	de71 e8 28	eorb	8,-x
ddd4c a8 b0	eora	1,sp+	de73 e8 68	eorb	8,-y
ddde a8 30	eora	1,x+	de75 e8 9f	eorb	-1,sp
dde0 a8 70	eora	1,y+	de77 e8 1f	eorb	-1,x
dde2 a8 81	eora	1,sp	de79 e8 5f	eorb	-1,y
dde4 a8 01	eora	1,x	de7b e8 90	eorb	-16,sp
dde6 a8 41	eora	1,y	de7d e8 10	eorb	-16,x
dde8 a8 bf	eora	1,sp-	de7f e8 50	eorb	-16,y
ddea a8 3f	eora	1,x-	de81 e8 f1 ef	eorb	-17,sp
ddec a8 7f	eora	1,y-	de84 e8 e1 ef	eorb	-17,x
ddee a8 f8 7d	eora	125,pc	de87 e8 e9 ef	eorb	-17,y
ddf1 a8 f0 7d	eora	125,sp	de8a e8 d2	eorb	-small,pc
ddf4 a8 e0 7d	eora	125,x	de8c e8 92	eorb	-small,sp
ddf7 a8 e8 7d	eora	125,y	de8e e8 12	eorb	-small,x
ddfa a8 8f	eora	15,sp	de90 e8 52	eorb	-small,y
ddfc a8 0f	eora	15,x	de92 e8 c0	eorb	0,pc
ddfe a8 4f	eora	15,y	de94 e8 80	eorb	0,sp
de00 a8 f0 10	eora	16,sp	de96 e8 00	eorb	0,x
de03 a8 e0 10	eora	16,x	de98 e8 40	eorb	0,y
de06 a8 e8 10	eora	16,y	de9a e8 b0	eorb	1,sp+
de09 a8 b7	eora	8,sp+	de9c e8 30	eorb	1,x+
de0b a8 37	eora	8,x+	de9e e8 70	eorb	1,y+
de0d a8 77	eora	8,y+	dea0 e8 81	eorb	1,sp
de0f a8 b8	eora	8,sp-	dea2 e8 01	eorb	1,x
de11 a8 38	eora	8,x-	dea4 e8 41	eorb	1,y
de13 a8 78	eora	8,y-	dea6 e8 bf	eorb	1,sp-
de15 a8 f4	eora	a,sp	dea8 e8 3f	eorb	1,x-
de17 a8 e4	eora	a,x	deaa e8 7f	eorb	1,y-
de19 a8 ec	eora	a,y	deac e8 f8 7d	eorb	125,pc
de1b a8 f5	eora	b,sp	deaf e8 f0 7d	eorb	125,sp
de1d a8 e5	eora	b,x	deb2 e8 e0 7d	eorb	125,x
de1f a8 ed	eora	b,y	deb5 e8 e8 7d	eorb	125,y
de21 a8 f6	eora	d,sp	deb8 e8 8f	eorb	15,sp
de23 a8 e6	eora	d,x	deba e8 0f	eorb	15,x
de25 a8 ee	eora	d,y	debc e8 4f	eorb	15,y
de27 98 55	eora	dir	debe e8 f0 10	eorb	16,sp
de29 98 55	eora	dir	dec1 e8 e0 10	eorb	16,x
de2b b8 01 88	eora	ext	dec4 e8 e8 10	eorb	16,y
de2e b8 01 88	eora	ext	dec7 e8 b7	eorb	8,sp+
de31 a8 f2 01 88	eora	ext,sp	dec9 e8 37	eorb	8,x+
de35 a8 e2 01 88	eora	ext,x	decb e8 77	eorb	8,y+
de39 a8 ea 01 88	eora	ext,y	decd e8 b8	eorb	8,sp-
de3d a8 f8 37	eora	ind,pc	decf e8 38	eorb	8,x-
de40 a8 f0 37	eora	ind,sp	ded1 e8 78	eorb	8,y-
de43 a8 e0 37	eora	ind,x	ded3 e8 f4	eorb	a,sp
de46 a8 e8 37	eora	ind,y	ded5 e8 e4	eorb	a,x
de49 a8 ce	eora	small,pc	ded7 e8 ec	eorb	a,y
de4b a8 8e	eora	small,sp	ded9 e8 f5	eorb	b,sp
de4d a8 0e	eora	small,x	dedb e8 e5	eorb	b,x
de4f a8 4e	eora	small,y	dedd e8 ed	eorb	b,y
de51 c8 72	eorb	#immed	dedf e8 f6	eorb	d,sp
de53 c8 72	eorb	#immed	dee1 e8 e6	eorb	d,x
de55 e8 a0	eorb	1,+sp	dee3 e8 ee	eorb	d,y
de57 e8 20	eorb	1,+x	dee5 d8 55	eorb	dir
de59 e8 60	eorb	1,+y	dee7 d8 55	eorb	dir
de5b e8 a7	eorb	8,+sp	dee9 f8 01 88	eorb	ext
de5d e8 27	eorb	8,+x	deec f8 01 88	eorb	ext



```

deef e8 f2 01 88 eorb ext,sp
def3 e8 e2 01 88 eorb ext,x
def7 e8 ea 01 88 eorb ext,y
defb e8 f8 37 eorb ind,pc
defe e8 f0 37 eorb ind,sp
df01 e8 e0 37 eorb ind,x
df04 e8 e8 37 eorb ind,y
df07 e8 ce eorb small,pc
df09 e8 8e eorb small,sp
df0b e8 0e eorb small,x
df0d e8 4e eorb small,y
df0f 18 3f 05 etbl 5++,x
df12 b7 80 exg a a
df14 b7 81 exg a b
df16 b7 81 exg a,b
df18 b7 82 exg a ccr
df1a b7 84 exg a d
df1c b7 87 exg a sp
df1e b7 85 exg a x
df20 b7 85 exg a,x
df22 b7 86 exg a y
df24 b7 85 exg a,x
df26 b7 90 exg b a
df28 b7 91 exg b b
df2a b7 92 exg b ccr
df2c b7 94 exg b d
df2e b7 97 exg b sp
df30 b7 95 exg b x
df32 b7 96 exg b y
df34 b7 a0 exg ccr a
df36 b7 a1 exg ccr b
df38 b7 a2 exg ccr ccr
df3a b7 a4 exg ccr d
df3c b7 a7 exg ccr sp
df3e b7 a5 exg ccr x
df40 b7 a6 exg ccr y
df42 b7 c0 exg d a
df44 b7 c1 exg d b
df46 b7 c2 exg d ccr
df48 b7 c4 exg d d
df4a b7 c7 exg d sp
df4c b7 c5 exg d x
df4e b7 c6 exg d y
df50 b7 f0 exg sp a
df52 b7 f1 exg sp b
df54 b7 f2 exg sp ccr
df56 b7 f4 exg sp d
df58 b7 f7 exg sp sp
df5a b7 f5 exg sp x
df5c b7 f6 exg sp y
df5e b7 d0 exg x a
df60 b7 d1 exg x b
df62 b7 d2 exg x ccr
df64 b7 d4 exg x d
df66 b7 d7 exg x sp
df68 b7 d5 exg x x
df6a b7 d6 exg x y
df6c b7 d6 exg x,y
df6e b7 e0 exg y a
df70 b7 e1 exg y b
df72 b7 e2 exg y ccr
df74 b7 e4 exg y d
df76 b7 e7 exg y sp
df78 b7 e5 exg y x
df7a b7 e6 exg y y
df7c 18 11 fddiv

```

```

df7e 18 10 idiv
df80 62 a0 inc 1,+sp
df82 62 20 inc 1,+x
df84 62 60 inc 1,+y
df86 62 a7 inc 8,+sp
df88 62 27 inc 8,+x
df8a 62 67 inc 8,+y
df8c 62 c0 inc ,pc
df8e 62 80 inc ,sp
df90 62 00 inc ,x
df92 62 40 inc ,y
df94 62 af inc 1,-sp
df96 62 2f inc 1,-x
df98 62 6f inc 1,-y
df9a 62 a8 inc 8,-sp
df9c 62 28 inc 8,-x
df9e 62 68 inc 8,-y
dfa0 62 9f inc -1,sp
dfa2 62 1f inc -1,x
dfa4 62 5f inc -1,y
dfa6 62 90 inc -16,sp
dfa8 62 10 inc -16,x
dfaa 62 50 inc -16,y
dfac 62 f1 ef inc -17,sp
dfaf 62 e1 ef inc -17,x
dfb2 62 e9 ef inc -17,y
dfb5 62 d2 inc -small,pc
dfb7 62 92 inc -small,sp
dfb9 62 12 inc -small,x
dfbb 62 52 inc -small,y
dfbd 62 c0 inc 0,pc
dfbf 62 80 inc 0,sp
dfc1 62 00 inc 0,x
dfc3 62 40 inc 0,y
dfc5 62 b0 inc 1,sp+
dfc7 62 30 inc 1,x+
dfc9 62 70 inc 1,y+
dfcb 62 81 inc 1,sp
dfcd 62 01 inc 1,x
dfcf 62 41 inc 1,y
dfd1 62 bf inc 1,sp-
dfd3 62 3f inc 1,x-
dfd5 62 7f inc 1,y-
dfd7 62 f8 7d inc 125,pc
dfda 62 f0 7d inc 125,sp
dfdd 62 e0 7d inc 125,x
dfe0 62 e8 7d inc 125,y
dfe3 62 8f inc 15,sp
dfe5 62 0f inc 15,x
dfe7 62 4f inc 15,y
dfe9 62 f0 10 inc 16,sp
dfec 62 e0 10 inc 16,x
dfef 62 e8 10 inc 16,y
dff2 62 b7 inc 8,sp+
dff4 62 37 inc 8,x+
dff6 62 77 inc 8,y+
dff8 62 b8 inc 8,sp-
dfffa 62 38 inc 8,x-
dfffc 62 78 inc 8,y-
dffe 62 f4 inc a,sp
e000 62 e4 inc a,x
e002 62 ec inc a,y
e004 62 f5 inc b,sp
e006 62 e5 inc b,x
e008 62 ed inc b,y
e00a 62 f6 inc d,sp

```

e00c 62 e6	inc	d,x	e09f 05 e8 7d	jmp	125,y
e00e 62 ee	inc	d,y	e0a2 05 8f	jmp	15,sp
e010 72 00 55	inc	dir	e0a4 05 0f	jmp	15,x
e013 72 01 88	inc	ext	e0a6 05 4f	jmp	15,y
e016 72 01 88	inc	ext	e0a8 05 f0 10	jmp	16,sp
e019 62 f2 01 88	inc	ext,sp	e0ab 05 e0 10	jmp	16,x
e01d 62 e2 01 88	inc	ext,x	e0ae 05 e8 10	jmp	16,y
e021 62 ea 01 88	inc	ext,y	e0b1 05 b7	jmp	8,sp+
e025 62 f8 37	inc	ind,pc	e0b3 05 37	jmp	8,x+
e028 62 f0 37	inc	ind,sp	e0b5 05 77	jmp	8,y+
e02b 62 e0 37	inc	ind,x	e0b7 05 b8	jmp	8,sp-
e02e 62 e8 37	inc	ind,y	e0b9 05 38	jmp	8,x-
e031 62 ce	inc	small,pc	e0bb 05 78	jmp	8,y-
e033 62 8e	inc	small,sp	e0bd 05 f4	jmp	a,sp
e035 62 0e	inc	small,x	e0bf 05 e4	jmp	a,x
e037 62 4e	inc	small,y	e0c1 05 ec	jmp	a,y
e039 42	inca		e0c3 05 f5	jmp	b,sp
e03a 52	incb		e0c5 05 e5	jmp	b,x
e03b 1b 81	ins		e0c7 05 ed	jmp	b,y
e03d 08	inx		e0c9 05 f6	jmp	d,sp
e03e 02	iny		e0cb 05 e6	jmp	d,x
e03f 05 a0	jmp	1,+sp	e0cd 05 ee	jmp	d,y
e041 05 20	jmp	1,+x	e0cf 06 00 55	jmp	dir
e043 05 60	jmp	1,+y	e0d2 06 01 88	jmp	ext
e045 05 a7	jmp	8,+sp	e0d5 06 01 88	jmp	ext
e047 05 27	jmp	8,+x	e0d8 05 f2 01 88	jmp	ext,sp
e049 05 67	jmp	8,+y	e0dc 05 e2 01 88	jmp	ext,x
e04b 05 c0	jmp	,pc	e0e0 05 ea 01 88	jmp	ext,y
e04d 05 80	jmp	,sp	e0e4 05 f8 37	jmp	ind,pc
e04f 05 00	jmp	,x	e0e7 05 f0 37	jmp	ind,sp
e051 05 40	jmp	,y	e0ea 05 e0 37	jmp	ind,x
e053 05 af	jmp	1,-sp	e0ed 05 e8 37	jmp	ind,y
e055 05 2f	jmp	1,-x	e0f0 05 ce	jmp	small,pc
e057 05 6f	jmp	1,-y	e0f2 05 8e	jmp	small,sp
e059 05 a8	jmp	8,-sp	e0f4 05 0e	jmp	small,x
e05b 05 28	jmp	8,-x	e0f6 05 4e	jmp	small,y
e05d 05 68	jmp	8,-y	e0f8 15 a0	jsr	1,+sp
e05f 05 9f	jmp	-1,sp	e0fa 15 20	jsr	1,+x
e061 05 1f	jmp	-1,x	e0fc 15 60	jsr	1,+y
e063 05 5f	jmp	-1,y	e0fe 15 a7	jsr	8,+sp
e065 05 90	jmp	-16,sp	e100 15 27	jsr	8,+x
e067 05 10	jmp	-16,x	e102 15 67	jsr	8,+y
e069 05 50	jmp	-16,y	e104 15 c0	jsr	,pc
e06b 05 f1 ef	jmp	-17,sp	e106 15 80	jsr	,sp
e06e 05 e1 ef	jmp	-17,x	e108 15 00	jsr	,x
e071 05 e9 ef	jmp	-17,y	e10a 15 40	jsr	,y
e074 05 d2	jmp	-small,pc	e10c 15 af	jsr	1,-sp
e076 05 92	jmp	-small,sp	e10e 15 2f	jsr	1,-x
e078 05 12	jmp	-small,x	e110 15 6f	jsr	1,-y
e07a 05 52	jmp	-small,y	e112 15 a8	jsr	8,-sp
e07c 05 c0	jmp	0,pc	e114 15 28	jsr	8,-x
e07e 05 80	jmp	0,sp	e116 15 68	jsr	8,-y
e080 05 00	jmp	0,x	e118 15 9f	jsr	-1,sp
e082 05 40	jmp	0,y	e11a 15 1f	jsr	-1,x
e084 05 b0	jmp	1,sp+	e11c 15 5f	jsr	-1,y
e086 05 30	jmp	1,x+	e11e 15 90	jsr	-16,sp
e088 05 70	jmp	1,y+	e120 15 10	jsr	-16,x
e08a 05 81	jmp	1,sp	e122 15 50	jsr	-16,y
e08c 05 01	jmp	1,x	e124 15 f1 ef	jsr	-17,sp
e08e 05 41	jmp	1,y	e127 15 e1 ef	jsr	-17,x
e090 05 bf	jmp	1,sp-	e12a 15 e9 ef	jsr	-17,y
e092 05 3f	jmp	1,x-	e12d 15 d2	jsr	-small,pc
e094 05 7f	jmp	1,y-	e12f 15 92	jsr	-small,sp
e096 05 f8 7d	jmp	125,pc	e131 15 12	jsr	-small,x
e099 05 f0 7d	jmp	125,sp	e133 15 52	jsr	-small,y
e09c 05 e0 7d	jmp	125,x	e135 15 c0	jsr	0,pc

e137 15 80	jsr	0,sp	e1e9 18 20 ff fc	lbra	*
e139 15 00	jsr	0,x	e1ed 18 21 ff fc	lbrn	*
e13b 15 40	jsr	0,y	e1f1 15 fa ff fc	lbsr	*
e13d 15 b0	jsr	1,sp+	e1f5 18 28 ff fc	lbvc	*
e13f 15 30	jsr	1,x+	e1f9 18 29 ff fc	lbvs	*
e141 15 70	jsr	1,y+	e1fd 86 72	ldaa	#immed
e143 15 81	jsr	1,sp	e1ff 86 72	ldaa	#immed
e145 15 01	jsr	1,x	e201 a6 a0	ldaa	1,+sp
e147 15 41	jsr	1,y	e203 a6 20	ldaa	1,+x
e149 15 bf	jsr	1,sp-	e205 a6 60	ldaa	1,+y
e14b 15 3f	jsr	1,x-	e207 a6 a7	ldaa	8,+sp
e14d 15 7f	jsr	1,y-	e209 a6 27	ldaa	8,+x
e14f 15 f8 7d	jsr	125,pc	e20b a6 67	ldaa	8,+y
e152 15 f0 7d	jsr	125,sp	e20d a6 c0	ldaa	,pc
e155 15 e0 7d	jsr	125,x	e20f a6 80	ldaa	,sp
e158 15 e8 7d	jsr	125,y	e211 a6 00	ldaa	,x
e15b 15 8f	jsr	15,sp	e213 a6 40	ldaa	,y
e15d 15 0f	jsr	15,x	e215 a6 af	ldaa	1,-sp
e15f 15 4f	jsr	15,y	e217 a6 2f	ldaa	1,-x
e161 15 f0 10	jsr	16,sp	e219 a6 6f	ldaa	1,-y
e164 15 e0 10	jsr	16,x	e21b a6 a8	ldaa	8,-sp
e167 15 e8 10	jsr	16,y	e21d a6 28	ldaa	8,-x
e16a 15 b7	jsr	8,sp+	e21f a6 68	ldaa	8,-y
e16c 15 37	jsr	8,x+	e221 a6 9f	ldaa	-1,sp
e16e 15 77	jsr	8,y+	e223 a6 1f	ldaa	-1,x
e170 15 b8	jsr	8,sp-	e225 a6 5f	ldaa	-1,y
e172 15 38	jsr	8,x-	e227 a6 90	ldaa	-16,sp
e174 15 78	jsr	8,y-	e229 a6 10	ldaa	-16,x
e176 15 f4	jsr	a,sp	e22b a6 50	ldaa	-16,y
e178 15 e4	jsr	a,x	e22d a6 f1 ef	ldaa	-17,sp
e17a 15 ec	jsr	a,y	e230 a6 e1 ef	ldaa	-17,x
e17c 15 f5	jsr	b,sp	e233 a6 e9 ef	ldaa	-17,y
e17e 15 e5	jsr	b,x	e236 a6 d2	ldaa	-small,pc
e180 15 ed	jsr	b,y	e238 a6 92	ldaa	-small,sp
e182 15 f6	jsr	d,sp	e23a a6 12	ldaa	-small,x
e184 15 e6	jsr	d,x	e23c a6 52	ldaa	-small,y
e186 15 ee	jsr	d,y	e23e a6 c0	ldaa	0,pc
e188 17 55	jsr	dir	e240 a6 80	ldaa	0,sp
e18a 17 55	jsr	dir	e242 a6 00	ldaa	0,x
e18c 16 01 88	jsr	ext	e244 a6 40	ldaa	0,y
e18f 16 01 88	jsr	ext	e246 a6 b0	ldaa	1,sp+
e192 16 01 88	jsr	ext	e248 a6 30	ldaa	1,x+
e195 15 f2 01 88	jsr	ext,sp	e24a a6 70	ldaa	1,y+
e199 15 e2 01 88	jsr	ext,x	e24c a6 81	ldaa	1,sp
e19d 15 ea 01 88	jsr	ext,y	e24e a6 01	ldaa	1,x
e1a1 15 f8 37	jsr	ind,pc	e250 a6 41	ldaa	1,y
e1a4 15 f0 37	jsr	ind,sp	e252 a6 bf	ldaa	1,sp-
e1a7 15 e0 37	jsr	ind,x	e254 a6 3f	ldaa	1,x-
e1aa 15 e8 37	jsr	ind,y	e256 a6 7f	ldaa	1,y-
e1ad 15 ce	jsr	small,pc	e258 a6 f8 7d	ldaa	125,pc
e1af 15 8e	jsr	small,sp	e25b a6 f0 7d	ldaa	125,sp
e1b1 15 0e	jsr	small,x	e25e a6 e0 7d	ldaa	125,x
e1b3 15 4e	jsr	small,y	e261 a6 e8 7d	ldaa	125,y
e1b5 18 24 ff fc	lbcc	*	e264 a6 8f	ldaa	15,sp
e1b9 18 24 ff fc	lbcc	*	e266 a6 0f	ldaa	15,x
e1bd 18 25 ff fc	lbcs	*	e268 a6 4f	ldaa	15,y
e1c1 18 27 ff fc	lbeq	*	e26a a6 f0 10	ldaa	16,sp
e1c5 18 2c ff fc	lbge	*	e26d a6 e0 10	ldaa	16,x
e1c9 18 2e ff fc	lbgt	*	e270 a6 e8 10	ldaa	16,y
e1cd 18 22 ff fc	lbhi	*	e273 a6 b7	ldaa	8,sp+
e1d1 18 2f ff fc	lble	*	e275 a6 37	ldaa	8,x+
e1d5 18 23 ff fc	bls	*	e277 a6 77	ldaa	8,y+
e1d9 18 2d ff fc	blt	*	e279 a6 b8	ldaa	8,sp-
e1dd 18 2b ff fc	lbmi	*	e27b a6 38	ldaa	8,x-
e1e1 18 26 ff fc	lbne	*	e27d a6 78	ldaa	8,y-
e1e5 18 2a ff fc	lbpl	*	e27f a6 f4	ldaa	a,sp

e281 a6 e4	ldaa a,x	e314 e6 7f	ldab 1,y-
e283 a6 ec	ldaa a,y	e316 e6 f8 7d	ldab 125,pc
e285 a6 f5	ldaa b,sp	e319 e6 f0 7d	ldab 125,sp
e287 a6 e5	ldaa b,x	e31c e6 e0 7d	ldab 125,x
e289 a6 ed	ldaa b,y	e31f e6 e8 7d	ldab 125,y
e28b a6 f6	ldaa d,sp	e322 e6 8f	ldab 15,sp
e28d a6 e6	ldaa d,x	e324 e6 0f	ldab 15,x
e28f a6 ee	ldaa d,y	e326 e6 4f	ldab 15,y
e291 96 55	ldaa dir	e328 e6 f0 10	ldab 16,sp
e293 96 55	ldaa dir	e32b e6 e0 10	ldab 16,x
e295 b6 01 88	ldaa ext	e32e e6 e8 10	ldab 16,y
e298 b6 01 88	ldaa ext	e331 e6 b7	ldab 8,sp+
e29b a6 f2 01 88	ldaa ext,sp	e333 e6 37	ldab 8,x+
e29f a6 e2 01 88	ldaa ext,x	e335 e6 77	ldab 8,y+
e2a3 a6 ea 01 88	ldaa ext,y	e337 e6 b8	ldab 8,sp-
e2a7 a6 f8 37	ldaa ind,pc	e339 e6 38	ldab 8,x-
e2aa a6 f0 37	ldaa ind,sp	e33b e6 78	ldab 8,y-
e2ad a6 e0 37	ldaa ind,x	e33d e6 f4	ldab a,sp
e2b0 a6 e8 37	ldaa ind,y	e33f e6 e4	ldab a,x
e2b3 a6 ce	ldaa small,pc	e341 e6 ec	ldab a,y
e2b5 a6 8e	ldaa small,sp	e343 e6 f5	ldab b,sp
e2b7 a6 0e	ldaa small,x	e345 e6 e5	ldab b,x
e2b9 a6 4e	ldaa small,y	e347 e6 ed	ldab b,y
e2bb c6 72	ldab #immed	e349 e6 f6	ldab d,sp
e2bd c6 72	ldab #immed	e34b e6 e6	ldab d,x
e2bf e6 a0	ldab 1,+sp	e34d e6 ee	ldab d,y
e2c1 e6 20	ldab 1,+x	e34f d6 55	ldab dir
e2c3 e6 60	ldab 1,+y	e351 d6 55	ldab dir
e2c5 e6 a7	ldab 8,+sp	e353 f6 01 88	ldab ext
e2c7 e6 27	ldab 8,+x	e356 f6 01 88	ldab ext
e2c9 e6 67	ldab 8,+y	e359 e6 f2 01 88	ldab ext,sp
e2cb e6 c0	ldab ,pc	e35d e6 e2 01 88	ldab ext,x
e2cd e6 80	ldab ,sp	e361 e6 ea 01 88	ldab ext,y
e2cf e6 00	ldab ,x	e365 e6 f8 37	ldab ind,pc
e2d1 e6 40	ldab ,y	e368 e6 f0 37	ldab ind,sp
e2d3 e6 af	ldab 1,-sp	e36b e6 e0 37	ldab ind,x
e2d5 e6 2f	ldab 1,-x	e36e e6 e8 37	ldab ind,y
e2d7 e6 6f	ldab 1,-y	e371 e6 ce	ldab small,pc
e2d9 e6 a8	ldab 8,-sp	e373 e6 8e	ldab small,sp
e2db e6 28	ldab 8,-x	e375 e6 0e	ldab small,x
e2dd e6 68	ldab 8,-y	e377 e6 4e	ldab small,y
e2df e6 9f	ldab -1,sp	e379 cc 00 72	ldd #immed
e2e1 e6 1f	ldab -1,x	e37c cc 00 72	ldd #immed
e2e3 e6 5f	ldab -1,y	e37f ec a0	ldd 1,+sp
e2e5 e6 90	ldab -16,sp	e381 ec 20	ldd 1,+x
e2e7 e6 10	ldab -16,x	e383 ec 60	ldd 1,+y
e2e9 e6 50	ldab -16,y	e385 ec a7	ldd 8,+sp
e2eb e6 f1 ef	ldab -17,sp	e387 ec 27	ldd 8,+x
e2ee e6 e1 ef	ldab -17,x	e389 ec 67	ldd 8,+y
e2f1 e6 e9 ef	ldab -17,y	e38b ec c0	ldd ,pc
e2f4 e6 d2	ldab -small,pc	e38d ec 80	ldd ,sp
e2f6 e6 92	ldab -small,sp	e38f ec 00	ldd ,x
e2f8 e6 12	ldab -small,x	e391 ec 40	ldd ,y
e2fa e6 52	ldab -small,y	e393 ec af	ldd 1,-sp
e2fc e6 c0	ldab 0,pc	e395 ec 2f	ldd 1,-x
e2fe e6 80	ldab 0,sp	e397 ec 6f	ldd 1,-y
e300 e6 00	ldab 0,x	e399 ec a8	ldd 8,-sp
e302 e6 40	ldab 0,y	e39b ec 28	ldd 8,-x
e304 e6 b0	ldab 1,sp+	e39d ec 68	ldd 8,-y
e306 e6 30	ldab 1,x+	e39f ec 9f	ldd -1,sp
e308 e6 70	ldab 1,y+	e3a1 ec 1f	ldd -1,x
e30a e6 81	ldab 1,sp	e3a3 ec 5f	ldd -1,y
e30c e6 01	ldab 1,x	e3a5 ec 90	ldd -16,sp
e30e e6 41	ldab 1,y	e3a7 ec 10	ldd -16,x
e310 e6 bf	ldab 1,sp-	e3a9 ec 50	ldd -16,y
e312 e6 3f	ldab 1,x-	e3ab ec f1 ef	ldd -17,sp

e3ae	ec	e1	ef	ldd	-17,x	e449	ef	67	lds	8,+y
e3b1	ec	e9	ef	ldd	-17,y	e44b	ef	c0	lds	,pc
e3b4	ec	d2		ldd	-small,pc	e44d	ef	80	lds	,sp
e3b6	ec	92		ldd	-small,sp	e44f	ef	00	lds	,x
e3b8	ec	12		ldd	-small,x	e451	ef	40	lds	,y
e3ba	ec	52		ldd	-small,y	e453	ef	af	lds	1,-sp
e3bc	ec	c0		ldd	0,pc	e455	ef	2f	lds	1,-x
e3be	ec	80		ldd	0,sp	e457	ef	6f	lds	1,-y
e3c0	ec	00		ldd	0,x	e459	ef	a8	lds	8,-sp
e3c2	ec	40		ldd	0,y	e45b	ef	28	lds	8,-x
e3c4	ec	b0		ldd	1,sp+	e45d	ef	68	lds	8,-y
e3c6	ec	30		ldd	1,x+	e45f	ef	9f	lds	-1,sp
e3c8	ec	70		ldd	1,y+	e461	ef	1f	lds	-1,x
e3ca	ec	81		ldd	1,sp	e463	ef	5f	lds	-1,y
e3cc	ec	01		ldd	1,x	e465	ef	90	lds	-16,sp
e3ce	ec	41		ldd	1,y	e467	ef	10	lds	-16,x
e3d0	ec	bf		ldd	1,sp-	e469	ef	50	lds	-16,y
e3d2	ec	3f		ldd	1,x-	e46b	ef	f1 ef	lds	-17,sp
e3d4	ec	7f		ldd	1,y-	e46e	ef	e1 ef	lds	-17,x
e3d6	ec	f8 7d		ldd	125,pc	e471	ef	e9 ef	lds	-17,y
e3d9	ec	f0 7d		ldd	125,sp	e474	ef	d2	lds	-small,pc
e3dc	ec	e0 7d		ldd	125,x	e476	ef	92	lds	-small,sp
e3df	ec	e8 7d		ldd	125,y	e478	ef	12	lds	-small,x
e3e2	ec	8f		ldd	15,sp	e47a	ef	52	lds	-small,y
e3e4	ec	0f		ldd	15,x	e47c	ef	c0	lds	0,pc
e3e6	ec	4f		ldd	15,y	e47e	ef	80	lds	0,sp
e3e8	ec	f0 10		ldd	16,sp	e480	ef	00	lds	0,x
e3eb	ec	e0 10		ldd	16,x	e482	ef	40	lds	0,y
e3ee	ec	e8 10		ldd	16,y	e484	ef	b0	lds	1,sp+
e3f1	ec	b7		ldd	8,sp+	e486	ef	30	lds	1,x+
e3f3	ec	37		ldd	8,x+	e488	ef	70	lds	1,y+
e3f5	ec	77		ldd	8,y+	e48a	ef	81	lds	1,sp
e3f7	ec	b8		ldd	8,sp-	e48c	ef	01	lds	1,x
e3f9	ec	38		ldd	8,x-	e48e	ef	41	lds	1,y
e3fb	ec	78		ldd	8,y-	e490	ef	bf	lds	1,sp-
e3fd	ec	f4		ldd	a,sp	e492	ef	3f	lds	1,x-
e3ff	ec	e4		ldd	a,x	e494	ef	7f	lds	1,y-
e401	ec	ec		ldd	a,y	e496	ef	f8 7d	lds	125,pc
e403	ec	f5		ldd	b,sp	e499	ef	f0 7d	lds	125,sp
e405	ec	e5		ldd	b,x	e49c	ef	e0 7d	lds	125,x
e407	ec	ed		ldd	b,y	e49f	ef	e8 7d	lds	125,y
e409	ec	f6		ldd	d,sp	e4a2	ef	8f	lds	15,sp
e40b	ec	e6		ldd	d,x	e4a4	ef	0f	lds	15,x
e40d	ec	ee		ldd	d,y	e4a6	ef	4f	lds	15,y
e40f	dc	55		ldd	dir	e4a8	ef	f0 10	lds	16,sp
e411	dc	55		ldd	dir	e4ab	ef	e0 10	lds	16,x
e413	fc	01 88		ldd	ext	e4ae	ef	e8 10	lds	16,y
e416	fc	01 88		ldd	ext	e4b1	ef	b7	lds	8,sp+
e419	ec	f2 01 88		ldd	ext,sp	e4b3	ef	37	lds	8,x+
e41d	ec	e2 01 88		ldd	ext,x	e4b5	ef	77	lds	8,y+
e421	ec	ea 01 88		ldd	ext,y	e4b7	ef	b8	lds	8,sp-
e425	ec	f8 37		ldd	ind,pc	e4b9	ef	38	lds	8,x-
e428	ec	f0 37		ldd	ind,sp	e4bb	ef	78	lds	8,y-
e42b	ec	e0 37		ldd	ind,x	e4bd	ef	f4	lds	a,sp
e42e	ec	e8 37		ldd	ind,y	e4bf	ef	e4	lds	a,x
e431	ec	ce		ldd	small,pc	e4c1	ef	ec	lds	a,y
e433	ec	8e		ldd	small,sp	e4c3	ef	f5	lds	b,sp
e435	ec	0e		ldd	small,x	e4c5	ef	e5	lds	b,x
e437	ec	4e		ldd	small,y	e4c7	ef	ed	lds	b,y
e439	cf	00 72		lds	#immed	e4c9	ef	f6	lds	d,sp
e43c	cf	00 72		lds	#immed	e4cb	ef	e6	lds	d,x
e43f	ef	a0		lds	1,+sp	e4cd	ef	ee	lds	d,y
e441	ef	20		lds	1,+x	e4cf	df	55	lds	dir
e443	ef	60		lds	1,+y	e4d1	ff	01 88	lds	ext
e445	ef	a7		lds	8,+sp	e4d4	ef	f2 01 88	lds	ext,sp
e447	ef	27		lds	8,+x	e4d8	ef	e2 01 88	lds	ext,x

e4dc ef ea 01 88	lds	ext,y	e572 ee b8	ldx	8,sp-
e4e0 ef f8 37	lds	ind,pc	e574 ee 38	ldx	8,x-
e4e3 ef f0 37	lds	ind,sp	e576 ee 78	ldx	8,y-
e4e6 ef e0 37	lds	ind,x	e578 ee f4	ldx	a,sp
e4e9 ef e8 37	lds	ind,y	e57a ee e4	ldx	a,x
e4ec ef ce	lds	small,pc	e57c ee ec	ldx	a,y
e4ee ef 8e	lds	small,sp	e57e ee f5	ldx	b,sp
e4f0 ef 0e	lds	small,x	e580 ee e5	ldx	b,x
e4f2 ef 4e	lds	small,y	e582 ee ed	ldx	b,y
e4f4 ce 00 72	ldx	#immed	e584 ee f6	ldx	d,sp
e4f7 ce 00 72	ldx	#immed	e586 ee e6	ldx	d,x
e4fa ee a0	ldx	1,+sp	e588 ee ee	ldx	d,y
e4fc ee 20	ldx	1,+x	e58a de 55	ldx	dir
e4fe ee 60	ldx	1,+y	e58c de 55	ldx	dir
e500 ee a7	ldx	8,+sp	e58e fe 01 88	ldx	ext
e502 ee 27	ldx	8,+x	e591 fe 01 88	ldx	ext
e504 ee 67	ldx	8,+y	e594 ee f2 01 88	ldx	ext,sp
e506 ee c0	ldx	,pc	e598 ee e2 01 88	ldx	ext,x
e508 ee 80	ldx	,sp	e59c ee ea 01 88	ldx	ext,y
e50a ee 00	ldx	,x	e5a0 ee f8 37	ldx	ind,pc
e50c ee 40	ldx	,y	e5a3 ee f0 37	ldx	ind,sp
e50e ee af	ldx	1,-sp	e5a6 ee e0 37	ldx	ind,x
e510 ee 2f	ldx	1,-x	e5a9 ee e8 37	ldx	ind,y
e512 ee 6f	ldx	1,-y	e5ac ee ce	ldx	small,pc
e514 ee a8	ldx	8,-sp	e5ae ee 8e	ldx	small,sp
e516 ee 28	ldx	8,-x	e5b0 ee 0e	ldx	small,x
e518 ee 68	ldx	8,-y	e5b2 ee 4e	ldx	small,y
e51a ee 9f	ldx	-1,sp	e5b4 cd 00 72	ldy	#immed
e51c ee 1f	ldx	-1,x	e5b7 cd 00 72	ldy	#immed
e51e ee 5f	ldx	-1,y	e5ba ed a0	ldy	1,+sp
e520 ee 90	ldx	-16,sp	e5bc ed 20	ldy	1,+x
e522 ee 10	ldx	-16,x	e5be ed 60	ldy	1,+y
e524 ee 50	ldx	-16,y	e5c0 ed a7	ldy	8,+sp
e526 ee f1 ef	ldx	-17,sp	e5c2 ed 27	ldy	8,+x
e529 ee e1 ef	ldx	-17,x	e5c4 ed 67	ldy	8,+y
e52c ee e9 ef	ldx	-17,y	e5c6 ed c0	ldy	,pc
e52f ee d2	ldx	-small,pc	e5c8 ed 80	ldy	,sp
e531 ee 92	ldx	-small,sp	e5ca ed 00	ldy	,x
e533 ee 12	ldx	-small,x	e5cc ed 40	ldy	,y
e535 ee 52	ldx	-small,y	e5ce ed af	ldy	1,-sp
e537 ee c0	ldx	0,pc	e5d0 ed 2f	ldy	1,-x
e539 ee 80	ldx	0,sp	e5d2 ed 6f	ldy	1,-y
e53b ee 00	ldx	0,x	e5d4 ed a8	ldy	8,-sp
e53d ee 40	ldx	0,y	e5d6 ed 28	ldy	8,-x
e53f ee b0	ldx	1,sp+	e5d8 ed 68	ldy	8,-y
e541 ee 30	ldx	1,x+	e5da ed 9f	ldy	-1,sp
e543 ee 70	ldx	1,y+	e5dc ed 1f	ldy	-1,x
e545 ee 81	ldx	1,sp	e5de ed 5f	ldy	-1,y
e547 ee 01	ldx	1,x	e5e0 ed 90	ldy	-16,sp
e549 ee 41	ldx	1,y	e5e2 ed 10	ldy	-16,x
e54b ee bf	ldx	1,sp-	e5e4 ed 50	ldy	-16,y
e54d ee 3f	ldx	1,x-	e5e6 ed f1 ef	ldy	-17,sp
e54f ee 7f	ldx	1,y-	e5e9 ed e1 ef	ldy	-17,x
e551 ee f8 7d	ldx	125,pc	e5ec ed e9 ef	ldy	-17,y
e554 ee f0 7d	ldx	125,sp	e5ef ed d2	ldy	-small,pc
e557 ee e0 7d	ldx	125,x	e5f1 ed 92	ldy	-small,sp
e55a ee e8 7d	ldx	125,y	e5f3 ed 12	ldy	-small,x
e55d ee 8f	ldx	15,sp	e5f5 ed 52	ldy	-small,y
e55f ee 0f	ldx	15,x	e5f7 ed c0	ldy	0,pc
e561 ee 4f	ldx	15,y	e5f9 ed 80	ldy	0,sp
e563 ee f0 10	ldx	16,sp	e5fb ed 00	ldy	0,x
e566 ee e0 10	ldx	16,x	e5fd ed 40	ldy	0,y
e569 ee e8 10	ldx	16,y	e5ff ed b0	ldy	1,sp+
e56c ee b7	ldx	8,sp+	e601 ed 30	ldy	1,x+
e56e ee 37	ldx	8,x+	e603 ed 70	ldy	1,y+
e570 ee 77	ldx	8,y+	e605 ed 81	ldy	1,sp



e607 ed 01	ldy	1,x	e69e 1b 50	leas	-16,y
e609 ed 41	ldy	1,y	e6a0 1b f1 ef	leas	-17,sp
e60b ed bf	ldy	1,sp-	e6a3 1b e1 ef	leas	-17,x
e60d ed 3f	ldy	1,x-	e6a6 1b e9 ef	leas	-17,y
e60f ed 7f	ldy	1,y-	e6a9 1b d2	leas	-small,pc
e611 ed f8 7d	ldy	125,pc	e6ab 1b 92	leas	-small,sp
e614 ed f0 7d	ldy	125,sp	e6ad 1b 12	leas	-small,x
e617 ed e0 7d	ldy	125,x	e6af 1b 52	leas	-small,y
e61a ed e8 7d	ldy	125,y	e6b1 1b c0	leas	0,pc
e61d ed 8f	ldy	15,sp	e6b3 1b 80	leas	0,sp
e61f ed 0f	ldy	15,x	e6b5 1b 00	leas	0,x
e621 ed 4f	ldy	15,y	e6b7 1b 40	leas	0,y
e623 ed f0 10	ldy	16,sp	e6b9 1b b0	leas	1,sp+
e626 ed e0 10	ldy	16,x	e6bb 1b 30	leas	1,x+
e629 ed e8 10	ldy	16,y	e6bd 1b 70	leas	1,y+
e62c ed b7	ldy	8,sp+	e6bf 1b 81	leas	1,sp
e62e ed 37	ldy	8,x+	e6c1 1b 01	leas	1,x
e630 ed 77	ldy	8,y+	e6c3 1b 41	leas	1,y
e632 ed b8	ldy	8,sp-	e6c5 1b bf	leas	1,sp-
e634 ed 38	ldy	8,x-	e6c7 1b 3f	leas	1,x-
e636 ed 78	ldy	8,y-	e6c9 1b 7f	leas	1,y-
e638 ed f4	ldy	a,sp	e6cb 1b f8 7d	leas	125,pc
e63a ed e4	ldy	a,x	e6ce 1b f0 7d	leas	125,sp
e63c ed ec	ldy	a,y	e6d1 1b e0 7d	leas	125,x
e63e ed f5	ldy	b,sp	e6d4 1b e8 7d	leas	125,y
e640 ed e5	ldy	b,x	e6d7 1b 8f	leas	15,sp
e642 ed ed	ldy	b,y	e6d9 1b 0f	leas	15,x
e644 ed f6	ldy	d,sp	e6db 1b 4f	leas	15,y
e646 ed e6	ldy	d,x	e6dd 1b f0 10	leas	16,sp
e648 ed ee	ldy	d,y	e6e0 1b e0 10	leas	16,x
e64a dd 55	ldy	dir	e6e3 1b e8 10	leas	16,y
e64c dd 55	ldy	dir	e6e6 1b b7	leas	8,sp+
e64e fd 01 88	ldy	ext	e6e8 1b 37	leas	8,x+
e651 fd 01 88	ldy	ext	e6ea 1b 77	leas	8,y+
e654 ed f2 01 88	ldy	ext,sp	e6ec 1b b8	leas	8,sp-
e658 ed e2 01 88	ldy	ext,x	e6ee 1b 38	leas	8,x-
e65c ed ea 01 88	ldy	ext,y	e6f0 1b 78	leas	8,y-
e660 ed f8 37	ldy	ind,pc	e6f2 1b f4	leas	a,sp
e663 ed f0 37	ldy	ind,sp	e6f4 1b e4	leas	a,x
e666 ed e0 37	ldy	ind,x	e6f6 1b ec	leas	a,y
e669 ed e8 37	ldy	ind,y	e6f8 1b f5	leas	b,sp
e66c ed ce	ldy	small,pc	e6fa 1b e5	leas	b,x
e66e ed 8e	ldy	small,sp	e6fc 1b ed	leas	b,y
e670 ed 0e	ldy	small,x	e6fe 1b f6	leas	d,sp
e672 ed 4e	ldy	small,y	e700 1b e6	leas	d,x
e674 1b a0	leas	1,+sp	e702 1b ee	leas	d,y
e676 1b 20	leas	1,+x	e704 1b f2 01 88	leas	ext,sp
e678 1b 60	leas	1,+y	e708 1b e2 01 88	leas	ext,x
e67a 1b a7	leas	8,+sp	e70c 1b ea 01 88	leas	ext,y
e67c 1b 27	leas	8,+x	e710 1b f8 37	leas	ind,pc
e67e 1b 67	leas	8,+y	e713 1b f0 37	leas	ind,sp
e680 1b c0	leas	,pc	e716 1b e0 37	leas	ind,x
e682 1b 80	leas	,sp	e719 1b e8 37	leas	ind,y
e684 1b 00	leas	,x	e71c 1b ce	leas	small,pc
e686 1b 40	leas	,y	e71e 1b 8e	leas	small,sp
e688 1b af	leas	1,-sp	e720 1b 0e	leas	small,x
e68a 1b 2f	leas	1,-x	e722 1b 4e	leas	small,y
e68c 1b 6f	leas	1,-y	e724 1a a0	leax	1,+sp
e68e 1b a8	leas	8,-sp	e726 1a 20	leax	1,+x
e690 1b 28	leas	8,-x	e728 1a 60	leax	1,+y
e692 1b 68	leas	8,-y	e72a 1a a7	leax	8,+sp
e694 1b 9f	leas	-1,sp	e72c 1a 27	leax	8,+x
e696 1b 1f	leas	-1,x	e72e 1a 67	leax	8,+y
e698 1b 5f	leas	-1,y	e730 1a c0	leax	,pc
e69a 1b 90	leas	-16,sp	e732 1a 80	leax	,sp
e69c 1b 10	leas	-16,x	e734 1a 00	leax	,x

e736 1a 40	leax	,y	e7ce 1a 8e	leax	small,sp
e738 1a af	leax	1,-sp	e7d0 1a 0e	leax	small,x
e73a 1a 2f	leax	1,-x	e7d2 1a 4e	leax	small,y
e73c 1a 6f	leax	1,-y	e7d4 19 a0	leay	1,+sp
e73e 1a a8	leax	8,-sp	e7d6 19 20	leay	1,+x
e740 1a 28	leax	8,-x	e7d8 19 60	leay	1,+y
e742 1a 68	leax	8,-y	e7da 19 a7	leay	8,+sp
e744 1a 9f	leax	-1,sp	e7dc 19 27	leay	8,+x
e746 1a 1f	leax	-1,x	e7de 19 67	leay	8,+y
e748 1a 5f	leax	-1,y	e7e0 19 c0	leay	,pc
e74a 1a 90	leax	-16,sp	e7e2 19 80	leay	,sp
e74c 1a 10	leax	-16,x	e7e4 19 00	leay	,x
e74e 1a 50	leax	-16,y	e7e6 19 40	leay	,y
e750 1a f1 ef	leax	-17,sp	e7e8 19 af	leay	1,-sp
e753 1a e1 ef	leax	-17,x	e7ea 19 2f	leay	1,-x
e756 1a e9 ef	leax	-17,y	e7ec 19 6f	leay	1,-y
e759 1a d2	leax	-small,pc	e7ee 19 a8	leay	8,-sp
e75b 1a 92	leax	-small,sp	e7f0 19 28	leay	8,-x
e75d 1a 12	leax	-small,x	e7f2 19 68	leay	8,-y
e75f 1a 52	leax	-small,y	e7f4 19 9f	leay	-1,sp
e761 1a c0	leax	0,pc	e7f6 19 1f	leay	-1,x
e763 1a 80	leax	0,sp	e7f8 19 5f	leay	-1,y
e765 1a 00	leax	0,x	e7fa 19 90	leay	-16,sp
e767 1a 40	leax	0,y	e7fc 19 10	leay	-16,x
e769 1a b0	leax	1,sp+	e7fe 19 50	leay	-16,y
e76b 1a 30	leax	1,x+	e800 19 f1 ef	leay	-17,sp
e76d 1a 70	leax	1,y+	e803 19 e1 ef	leay	-17,x
e76f 1a 81	leax	1,sp	e806 19 e9 ef	leay	-17,y
e771 1a 01	leax	1,x	e809 19 d2	leay	-small,pc
e773 1a 41	leax	1,y	e80b 19 92	leay	-small,sp
e775 1a bf	leax	1,sp-	e80d 19 12	leay	-small,x
e777 1a 3f	leax	1,x-	e80f 19 52	leay	-small,y
e779 1a 7f	leax	1,y-	e811 19 c0	leay	0,pc
e77b 1a f8 7d	leax	125,pc	e813 19 80	leay	0,sp
e77e 1a f0 7d	leax	125,sp	e815 19 00	leay	0,x
e781 1a e0 7d	leax	125,x	e817 19 40	leay	0,y
e784 1a e8 7d	leax	125,y	e819 19 b0	leay	1,sp+
e787 1a 8f	leax	15,sp	e81b 19 30	leay	1,x+
e789 1a 0f	leax	15,x	e81d 19 70	leay	1,y+
e78b 1a 4f	leax	15,y	e81f 19 81	leay	1,sp
e78d 1a f0 10	leax	16,sp	e821 19 01	leay	1,x
e790 1a e0 10	leax	16,x	e823 19 41	leay	1,y
e793 1a e8 10	leax	16,y	e825 19 bf	leay	1,sp-
e796 1a b7	leax	8,sp+	e827 19 3f	leay	1,x-
e798 1a 37	leax	8,x+	e829 19 7f	leay	1,y-
e79a 1a 77	leax	8,y+	e82b 19 f8 7d	leay	125,pc
e79c 1a b8	leax	8,sp-	e82e 19 f0 7d	leay	125,sp
e79e 1a 38	leax	8,x-	e831 19 e0 7d	leay	125,x
e7a0 1a 78	leax	8,y-	e834 19 e8 7d	leay	125,y
e7a2 1a f4	leax	a,sp	e837 19 8f	leay	15,sp
e7a4 1a e4	leax	a,x	e839 19 0f	leay	15,x
e7a6 1a ec	leax	a,y	e83b 19 4f	leay	15,y
e7a8 1a f5	leax	b,sp	e83d 19 f0 10	leay	16,sp
e7aa 1a e5	leax	b,x	e840 19 e0 10	leay	16,x
e7ac 1a ed	leax	b,y	e843 19 e8 10	leay	16,y
e7ae 1a f6	leax	d,sp	e846 19 b7	leay	8,sp+
e7b0 1a e6	leax	d,x	e848 19 37	leay	8,x+
e7b2 1a ee	leax	d,y	e84a 19 77	leay	8,y+
e7b4 1a f2 01 88	leax	ext,sp	e84c 19 b8	leay	8,sp-
e7b8 1a e2 01 88	leax	ext,x	e84e 19 38	leay	8,x-
e7bc 1a ea 01 88	leax	ext,y	e850 19 78	leay	8,y-
e7c0 1a f8 37	leax	ind,pc	e852 19 f4	leay	a,sp
e7c3 1a f0 37	leax	ind,sp	e854 19 e4	leay	a,x
e7c6 1a e0 37	leax	ind,x	e856 19 ec	leay	a,y
e7c9 1a e8 37	leax	ind,y	e858 19 f5	leay	b,sp
e7cc 1a ce	leax	small,pc	e85a 19 e5	leay	b,x



e85c 19 ed	leay	b,y	e8f3 68 e8 10	lsl	16,y
e85e 19 f6	leay	d,sp	e8f6 68 b7	lsl	8,sp+
e860 19 e6	leay	d,x	e8f8 68 37	lsl	8,x+
e862 19 ee	leay	d,y	e8fa 68 77	lsl	8,y+
e864 19 f2 01 88	leay	ext,sp	e8fc 68 b8	lsl	8,sp-
e868 19 e2 01 88	leay	ext,x	e8fe 68 38	lsl	8,x-
e86c 19 ea 01 88	leay	ext,y	e900 68 78	lsl	8,y-
e870 19 f8 37	leay	ind,pc	e902 68 f4	lsl	a,sp
e873 19 f0 37	leay	ind,sp	e904 68 e4	lsl	a,x
e876 19 e0 37	leay	ind,x	e906 68 ec	lsl	a,y
e879 19 e8 37	leay	ind,y	e908 68 f5	lsl	b,sp
e87c 19 ce	leay	small,pc	e90a 68 e5	lsl	b,x
e87e 19 8e	leay	small,sp	e90c 68 ed	lsl	b,y
e880 19 0e	leay	small,x	e90e 68 f6	lsl	d,sp
e882 19 4e	leay	small,y	e910 68 e6	lsl	d,x
e884 68 a0	lsl	1,+sp	e912 68 ee	lsl	d,y
e886 68 20	lsl	1,+x	e914 78 00 55	lsl	dir
e888 68 60	lsl	1,+y	e917 78 01 88	lsl	ext
e88a 68 a7	lsl	8,+sp	e91a 78 01 88	lsl	ext
e88c 68 27	lsl	8,+x	e91d 68 f2 01 88	lsl	ext,sp
e88e 68 67	lsl	8,+y	e921 68 e2 01 88	lsl	ext,x
e890 68 c0	lsl	,pc	e925 68 ea 01 88	lsl	ext,y
e892 68 80	lsl	,sp	e929 68 f8 37	lsl	ind,pc
e894 68 00	lsl	,x	e92c 68 f0 37	lsl	ind,sp
e896 68 40	lsl	,y	e92f 68 e0 37	lsl	ind,x
e898 68 af	lsl	1,-sp	e932 68 e8 37	lsl	ind,y
e89a 68 2f	lsl	1,-x	e935 68 ce	lsl	small,pc
e89c 68 6f	lsl	1,-y	e937 68 8e	lsl	small,sp
e89e 68 a8	lsl	8,-sp	e939 68 0e	lsl	small,x
e8a0 68 28	lsl	8,-x	e93b 68 4e	lsl	small,y
e8a2 68 68	lsl	8,-y	e93d 48	lsla	
e8a4 68 9f	lsl	-1,sp	e93e 58	lslb	
e8a6 68 1f	lsl	-1,x	e93f 59	lsl d	
e8a8 68 5f	lsl	-1,y	e940 64 a0	lsr	1,+sp
e8aa 68 90	lsl	-16,sp	e942 64 20	lsr	1,+x
e8ac 68 10	lsl	-16,x	e944 64 60	lsr	1,+y
e8ae 68 50	lsl	-16,y	e946 64 a7	lsr	8,+sp
e8b0 68 f1 ef	lsl	-17,sp	e948 64 27	lsr	8,+x
e8b3 68 e1 ef	lsl	-17,x	e94a 64 67	lsr	8,+y
e8b6 68 e9 ef	lsl	-17,y	e94c 64 c0	lsr	,pc
e8b9 68 d2	lsl	-small,pc	e94e 64 80	lsr	,sp
e8bb 68 92	lsl	-small,sp	e950 64 00	lsr	,x
e8bd 68 12	lsl	-small,x	e952 64 40	lsr	,y
e8bf 68 52	lsl	-small,y	e954 64 af	lsr	1,-sp
e8c1 68 c0	lsl	0,pc	e956 64 2f	lsr	1,-x
e8c3 68 80	lsl	0,sp	e958 64 6f	lsr	1,-y
e8c5 68 00	lsl	0,x	e95a 64 a8	lsr	8,-sp
e8c7 68 40	lsl	0,y	e95c 64 28	lsr	8,-x
e8c9 68 b0	lsl	1,sp+	e95e 64 68	lsr	8,-y
e8cb 68 30	lsl	1,x+	e960 64 9f	lsr	-1,sp
e8cd 68 70	lsl	1,y+	e962 64 1f	lsr	-1,x
e8cf 68 81	lsl	1,sp	e964 64 5f	lsr	-1,y
e8d1 68 01	lsl	1,x	e966 64 90	lsr	-16,sp
e8d3 68 41	lsl	1,y	e968 64 10	lsr	-16,x
e8d5 68 bf	lsl	1,sp-	e96a 64 50	lsr	-16,y
e8d7 68 3f	lsl	1,x-	e96c 64 f1 ef	lsr	-17,sp
e8d9 68 7f	lsl	1,y-	e96f 64 e1 ef	lsr	-17,x
e8db 68 f8 7d	lsl	125,pc	e972 64 e9 ef	lsr	-17,y
e8de 68 f0 7d	lsl	125,sp	e975 64 d2	lsr	-small,pc
e8e1 68 e0 7d	lsl	125,x	e977 64 92	lsr	-small,sp
e8e4 68 e8 7d	lsl	125,y	e979 64 12	lsr	-small,x
e8e7 68 8f	lsl	15,sp	e97b 64 52	lsr	-small,y
e8e9 68 0f	lsl	15,x	e97d 64 c0	lsr	0,pc
e8eb 68 4f	lsl	15,y	e97f 64 80	lsr	0,sp
e8ed 68 f0 10	lsl	16,sp	e981 64 00	lsr	0,x
e8f0 68 e0 10	lsl	16,x	e983 64 40	lsr	0,y

e985 64 b0	lsr	1,sp+	ea27 18 18 28	maxa	8,-x
e987 64 30	lsr	1,x+	ea2a 18 18 68	maxa	8,-y
e989 64 70	lsr	1,y+	ea2d 18 18 9f	maxa	-1,sp
e98b 64 81	lsr	1,sp	ea30 18 18 1f	maxa	-1,x
e98d 64 01	lsr	1,x	ea33 18 18 5f	maxa	-1,y
e98f 64 41	lsr	1,y	ea36 18 18 90	maxa	-16,sp
e991 64 bf	lsr	1,sp-	ea39 18 18 10	maxa	-16,x
e993 64 3f	lsr	1,x-	ea3c 18 18 50	maxa	-16,y
e995 64 7f	lsr	1,y-	ea3f 18 18 f1 ef	maxa	-17,sp
e997 64 f8 7d	lsr	125,pc	ea43 18 18 e1 ef	maxa	-17,x
e99a 64 f0 7d	lsr	125,sp	ea47 18 18 e9 ef	maxa	-17,y
e99d 64 e0 7d	lsr	125,x	ea4b 18 18 d2	maxa	-small,pc
e9a0 64 e8 7d	lsr	125,y	ea4e 18 18 92	maxa	-small,sp
e9a3 64 8f	lsr	15,sp	ea51 18 18 12	maxa	-small,x
e9a5 64 0f	lsr	15,x	ea54 18 18 52	maxa	-small,y
e9a7 64 4f	lsr	15,y	ea57 18 18 c0	maxa	0,pc
e9a9 64 f0 10	lsr	16,sp	ea5a 18 18 80	maxa	0,sp
e9ac 64 e0 10	lsr	16,x	ea5d 18 18 00	maxa	0,x
e9af 64 e8 10	lsr	16,y	ea60 18 18 40	maxa	0,y
e9b2 64 b7	lsr	8,sp+	ea63 18 18 b0	maxa	1,sp+
e9b4 64 37	lsr	8,x+	ea66 18 18 30	maxa	1,x+
e9b6 64 77	lsr	8,y+	ea69 18 18 70	maxa	1,y+
e9b8 64 b8	lsr	8,sp-	ea6c 18 18 81	maxa	1,sp
e9ba 64 38	lsr	8,x-	ea6f 18 18 01	maxa	1,x
e9bc 64 78	lsr	8,y-	ea72 18 18 41	maxa	1,y
e9be 64 f4	lsr	a,sp	ea75 18 18 bf	maxa	1,sp-
e9c0 64 e4	lsr	a,x	ea78 18 18 3f	maxa	1,x-
e9c2 64 ec	lsr	a,y	ea7b 18 18 7f	maxa	1,y-
e9c4 64 f5	lsr	b,sp	ea7e 18 18 f8 7d	maxa	125,pc
e9c6 64 e5	lsr	b,x	ea82 18 18 f0 7d	maxa	125,sp
e9c8 64 ed	lsr	b,y	ea86 18 18 e0 7d	maxa	125,x
e9ca 64 f6	lsr	d,sp	ea8a 18 18 e8 7d	maxa	125,y
e9cc 64 e6	lsr	d,x	ea8e 18 18 8f	maxa	15,sp
e9ce 64 ee	lsr	d,y	ea91 18 18 0f	maxa	15,x
e9d0 74 00 55	lsr	dir	ea94 18 18 4f	maxa	15,y
e9d3 74 01 88	lsr	ext	ea97 18 18 f0 10	maxa	16,sp
e9d6 74 01 88	lsr	ext	ea9b 18 18 e0 10	maxa	16,x
e9d9 64 f2 01 88	lsr	ext,sp	ea9f 18 18 e8 10	maxa	16,y
e9dd 64 e2 01 88	lsr	ext,x	eaab 18 18 b7	maxa	8,sp+
e9e1 64 ea 01 88	lsr	ext,y	eaac 18 18 37	maxa	8,x+
e9e5 64 f8 37	lsr	ind,pc	eaad 18 18 77	maxa	8,y+
e9e8 64 f0 37	lsr	ind,sp	eaaf 18 18 b8	maxa	8,sp-
e9eb 64 e0 37	lsr	ind,x	eaaf 18 18 38	maxa	8,x-
e9ee 64 e8 37	lsr	ind,y	eaab 18 18 78	maxa	8,y-
e9f1 64 ce	lsr	small,pc	eaab 18 18 f4	maxa	a,sp
e9f3 64 8e	lsr	small,sp	eaab 18 18 e4	maxa	a,x
e9f5 64 0e	lsr	small,x	eaab 18 18 ec	maxa	a,y
e9f7 64 4e	lsr	small,y	eaab 18 18 f5	maxa	b,sp
e9f9 44	lsra		eac1 18 18 e5	maxa	b,x
e9fa 54	lsrb		eac4 18 18 ed	maxa	b,y
e9fb 49	lsrd		eac7 18 18 f6	maxa	d,sp
e9fc 49	lsrd		eaca 18 18 e6	maxa	d,x
e9fd 18 18 a0	maxa	1,+sp	eacd 18 18 ee	maxa	d,y
ea00 18 18 20	maxa	1,+x	ead0 18 18 f2 01 88	maxa	ext,sp
ea03 18 18 60	maxa	1,+y	ead5 18 18 e2 01 88	maxa	ext,x
ea06 18 18 a7	maxa	8,+sp	ead8 18 18 ea 01 88	maxa	ext,y
ea09 18 18 27	maxa	8,+x	eadf 18 18 f8 37	maxa	ind,pc
ea0c 18 18 67	maxa	8,+y	eaef 18 18 f0 37	maxa	ind,sp
ea0f 18 18 c0	maxa	,pc	eaef 18 18 e0 37	maxa	ind,x
ea12 18 18 80	maxa	,sp	eaef 18 18 e8 37	maxa	ind,y
ea15 18 18 00	maxa	,x	eaef 18 18 ce	maxa	small,pc
ea18 18 18 40	maxa	,y	eaf2 18 18 8e	maxa	small,sp
ea1b 18 18 af	maxa	1,-sp	eaf5 18 18 0e	maxa	small,x
ea1e 18 18 2f	maxa	1,-x	eaf8 18 18 4e	maxa	small,y
ea21 18 18 6f	maxa	1,-y	eafb 18 1c a0	maxm	1,+sp
ea24 18 18 a8	maxa	8,-sp	eafe 18 1c 20	maxm	1,+x

eb01	18	1c	60	maxm	1,+y	ebd3	18	1c	e2 01 88	maxm	ext,x
eb04	18	1c	a7	maxm	8,+sp	ebd8	18	1c	ea 01 88	maxm	ext,y
eb07	18	1c	27	maxm	8,+x	ebdd	18	1c	f8 37	maxm	ind,pc
eb0a	18	1c	67	maxm	8,+y	ebe1	18	1c	f0 37	maxm	ind,sp
eb0d	18	1c	c0	maxm	,pc	ebe5	18	1c	e0 37	maxm	ind,x
eb10	18	1c	80	maxm	,sp	ebe9	18	1c	e8 37	maxm	ind,y
eb13	18	1c	00	maxm	,x	ebed	18	1c	ce	maxm	small,pc
eb16	18	1c	40	maxm	,y	ebf0	18	1c	8e	maxm	small,sp
eb19	18	1c	af	maxm	1,-sp	ebf3	18	1c	0e	maxm	small,x
eb1c	18	1c	2f	maxm	1,-x	ebf6	18	1c	4e	maxm	small,y
eb1f	18	1c	6f	maxm	1,-y	ebf9	01			mem	
eb22	18	1c	a8	maxm	8,-sp	ebfa	18	19	a0	mina	1,+sp
eb25	18	1c	28	maxm	8,-x	ebfd	18	19	20	mina	1,+x
eb28	18	1c	68	maxm	8,-y	ec00	18	19	60	mina	1,+y
eb2b	18	1c	9f	maxm	-1,sp	ec03	18	19	a7	mina	8,+sp
eb2e	18	1c	1f	maxm	-1,x	ec06	18	19	27	mina	8,+x
eb31	18	1c	5f	maxm	-1,y	ec09	18	19	67	mina	8,+y
eb34	18	1c	90	maxm	-16,sp	ec0c	18	19	c0	mina	,pc
eb37	18	1c	10	maxm	-16,x	ec0f	18	19	80	mina	,sp
eb3a	18	1c	50	maxm	-16,y	ec12	18	19	00	mina	,x
eb3d	18	1c	f1 ef	maxm	-17,sp	ec15	18	19	40	mina	,y
eb41	18	1c	e1 ef	maxm	-17,x	ec18	18	19	af	mina	1,-sp
eb45	18	1c	e9 ef	maxm	-17,y	ec1b	18	19	2f	mina	1,-x
eb49	18	1c	d2	maxm	-small,pc	ec1e	18	19	6f	mina	1,-y
eb4c	18	1c	92	maxm	-small,sp	ec21	18	19	a8	mina	8,-sp
eb4f	18	1c	12	maxm	-small,x	ec24	18	19	28	mina	8,-x
eb52	18	1c	52	maxm	-small,y	ec27	18	19	68	mina	8,-y
eb55	18	1c	c0	maxm	0,pc	ec2a	18	19	9f	mina	-1,sp
eb58	18	1c	80	maxm	0,sp	ec2d	18	19	1f	mina	-1,x
eb5b	18	1c	00	maxm	0,x	ec30	18	19	5f	mina	-1,y
eb5e	18	1c	40	maxm	0,y	ec33	18	19	90	mina	-16,sp
eb61	18	1c	b0	maxm	1,sp+	ec36	18	19	10	mina	-16,x
eb64	18	1c	30	maxm	1,x+	ec39	18	19	50	mina	-16,y
eb67	18	1c	70	maxm	1,y+	ec3c	18	19	f1 ef	mina	-17,sp
eb6a	18	1c	81	maxm	1,sp	ec40	18	19	e1 ef	mina	-17,x
eb6d	18	1c	01	maxm	1,x	ec44	18	19	e9 ef	mina	-17,y
eb70	18	1c	41	maxm	1,y	ec48	18	19	d2	mina	-small,pc
eb73	18	1c	bf	maxm	1,sp-	ec4b	18	19	92	mina	-small,sp
eb76	18	1c	3f	maxm	1,x-	ec4e	18	19	12	mina	-small,x
eb79	18	1c	7f	maxm	1,y-	ec51	18	19	52	mina	-small,y
eb7c	18	1c	f8 7d	maxm	125,pc	ec54	18	19	c0	mina	0,pc
eb80	18	1c	f0 7d	maxm	125,sp	ec57	18	19	80	mina	0,sp
eb84	18	1c	e0 7d	maxm	125,x	ec5a	18	19	00	mina	0,x
eb88	18	1c	e8 7d	maxm	125,y	ec5d	18	19	40	mina	0,y
eb8c	18	1c	8f	maxm	15,sp	ec60	18	19	b0	mina	1,sp+
eb8f	18	1c	0f	maxm	15,x	ec63	18	19	30	mina	1,x+
eb92	18	1c	4f	maxm	15,y	ec66	18	19	70	mina	1,y+
eb95	18	1c	f0 10	maxm	16,sp	ec69	18	19	81	mina	1,sp
eb99	18	1c	e0 10	maxm	16,x	ec6c	18	19	01	mina	1,x
eb9d	18	1c	e8 10	maxm	16,y	ec6f	18	19	41	mina	1,y
eba1	18	1c	b7	maxm	8,sp+	ec72	18	19	bf	mina	1,sp-
eba4	18	1c	37	maxm	8,x+	ec75	18	19	3f	mina	1,x-
eba7	18	1c	77	maxm	8,y+	ec78	18	19	7f	mina	1,y-
ebaa	18	1c	b8	maxm	8,sp-	ec7b	18	19	f8 7d	mina	125,pc
ebad	18	1c	38	maxm	8,x-	ec7f	18	19	f0 7d	mina	125,sp
ebb0	18	1c	78	maxm	8,y-	ec83	18	19	e0 7d	mina	125,x
ebb3	18	1c	f4	maxm	a,sp	ec87	18	19	e8 7d	mina	125,y
ebb6	18	1c	e4	maxm	a,x	ec8b	18	19	8f	mina	15,sp
ebb9	18	1c	ec	maxm	a,y	ec8e	18	19	0f	mina	15,x
ebbc	18	1c	f5	maxm	b,sp	ec91	18	19	4f	mina	15,y
ebbf	18	1c	e5	maxm	b,x	ec94	18	19	f0 10	mina	16,sp
ebc2	18	1c	ed	maxm	b,y	ec98	18	19	e0 10	mina	16,x
ebc5	18	1c	f6	maxm	d,sp	ec9c	18	19	e8 10	mina	16,y
ebc8	18	1c	e6	maxm	d,x	eca0	18	19	b7	mina	8,sp+
ebcb	18	1c	ee	maxm	d,y	eca3	18	19	37	mina	8,x+
ebce	18	1c	f2 01 88	maxm	ext,sp	eca6	18	19	77	mina	8,y+

```
; funny ` test
; stinky `000 test
;
```

ee55	18	08	6b	72	movb	#immed 5,-y	ef61	18	0a	22	f5	movb	3,+x b,sp
ee59	18	08	85	72	movb	#immed 5,sp	ef65	18	0a	22	e5	movb	3,+x b,x
ee5d	18	0b	72	01 88	movb	#immed ext	ef69	18	0a	22	ed	movb	3,+x b,y
ee62	18	0a	a0	22	movb	1,+sp 3,+x	ef6d	18	0a	22	f6	movb	3,+x d,sp
ee66	18	0a	a0	6b	movb	1,+sp 5,-y	ef71	18	0a	22	e6	movb	3,+x d,x
ee6a	18	0a	a0	85	movb	1,+sp 5,sp	ef75	18	0a	22	ee	movb	3,+x d,y
ee6e	18	0d	a0	01 88	movb	1,+sp ext	ef79	18	0d	22	01 88	movb	3,+x ext
ee73	18	0a	20	22	movb	1,+x 3,+x	ef7e	18	0a	22	ce	movb	3,+x small,pc
ee77	18	0a	20	6b	movb	1,+x 5,-y	ef82	18	0a	22	8e	movb	3,+x small,sp
ee7b	18	0a	20	85	movb	1,+x 5,sp	ef86	18	0a	22	0e	movb	3,+x small,x
ee7f	18	0d	20	01 88	movb	1,+x ext	ef8a	18	0a	22	4e	movb	3,+x small,y
ee84	18	0a	60	22	movb	1,+y 3,+x	ef8e	18	0a	a7	22	movb	8,+sp 3,+x
ee88	18	0a	60	6b	movb	1,+y 5,-y	ef92	18	0a	a7	6b	movb	8,+sp 5,-y
ee8c	18	0a	60	85	movb	1,+y 5,sp	ef96	18	0a	a7	85	movb	8,+sp 5,sp
ee90	18	0d	60	01 88	movb	1,+y ext	ef9a	18	0d	a7	01 88	movb	8,+sp ext
ee95	18	0a	22	a0	movb	3,+x 1,+sp	ef9f	18	0a	27	22	movb	8,+x 3,+x
ee99	18	0a	22	20	movb	3,+x 1,+x	efa3	18	0a	27	6b	movb	8,+x 5,-y
ee9d	18	0a	22	60	movb	3,+x 1,+y	efa7	18	0a	27	85	movb	8,+x 5,sp
eea1	18	0a	22	a7	movb	3,+x 8,+sp	efab	18	0d	27	01 88	movb	8,+x ext
eea5	18	0a	22	27	movb	3,+x 8,+x	efb0	18	0a	67	22	movb	8,+y 3,+x
eea9	18	0a	22	67	movb	3,+x 8,+y	efb4	18	0a	67	6b	movb	8,+y 5,-y
eead	18	0a	22	c0	movb	3,+x ,pc	efb8	18	0a	67	85	movb	8,+y 5,sp
eeb1	18	0a	22	80	movb	3,+x ,sp	efbc	18	0d	67	01 88	movb	8,+y ext
eeb5	18	0a	22	00	movb	3,+x ,x	efc1	18	0a	c0	22	movb	,pc 3,+x
eeb9	18	0a	22	40	movb	3,+x ,y	efc5	18	0a	c0	6b	movb	,pc 5,-y
eebd	18	0a	22	af	movb	3,+x 1,-sp	efc9	18	0a	c0	85	movb	,pc 5,sp
eec1	18	0a	22	2f	movb	3,+x 1,-x	efcd	18	09	c0	00 00	movb	,pc ext
eec5	18	0a	22	6f	movb	3,+x 1,-y	efd2	18	0a	80	22	movb	,sp 3,+x
eec9	18	0a	22	a8	movb	3,+x 8,-sp	efd6	18	0a	80	6b	movb	,sp 5,-y
eecl	18	0a	22	28	movb	3,+x 8,-x	efda	18	0a	80	85	movb	,sp 5,sp
eed1	18	0a	22	68	movb	3,+x 8,-y	efde	18	09	80	00 00	movb	,sp ext
eed5	18	0a	22	9f	movb	3,+x -1,sp	efe3	18	0a	00	22	movb	,x 3,+x
eed9	18	0a	22	1f	movb	3,+x -1,x	efe7	18	0a	00	6b	movb	,x 5,-y
eedd	18	0a	22	5f	movb	3,+x -1,y	efeb	18	0a	00	85	movb	,x 5,sp
eee1	18	0a	22	90	movb	3,+x -16,sp	efef	18	09	00	00 00	movb	,x ext
eee5	18	0a	22	10	movb	3,+x -16,x	eff4	18	0a	40	22	movb	,y 3,+x
eee9	18	0a	22	50	movb	3,+x -16,y	eff8	18	0a	40	6b	movb	,y 5,-y
eeed	18	0a	22	d2	movb	3,+x -small,pc	effc	18	0a	40	85	movb	,y 5,sp
eef1	18	0a	22	92	movb	3,+x -small,sp	f000	18	09	40	00 00	movb	,y ext
eef5	18	0a	22	12	movb	3,+x -small,x	f005	18	0a	af	22	movb	1,-sp 3,+x
eef9	18	0a	22	52	movb	3,+x -small,y	f009	18	0a	af	6b	movb	1,-sp 5,-y
eefd	18	0a	22	c0	movb	3,+x 0,pc	f00d	18	0a	af	85	movb	1,-sp 5,sp
ef01	18	0a	22	80	movb	3,+x 0,sp	f011	18	0d	af	01 88	movb	1,-sp ext
ef05	18	0a	22	00	movb	3,+x 0,x	f016	18	0a	2f	22	movb	1,-x 3,+x
ef09	18	0a	22	40	movb	3,+x 0,y	f01a	18	0a	2f	6b	movb	1,-x 5,-y
ef0d	18	0a	22	b0	movb	3,+x 1,sp+	f01e	18	0a	2f	85	movb	1,-x 5,sp
ef11	18	0a	22	30	movb	3,+x 1,x+	f022	18	0d	2f	01 88	movb	1,-x ext
ef15	18	0a	22	70	movb	3,+x 1,y+	f027	18	0a	6f	22	movb	1,-y 3,+x
ef19	18	0a	22	81	movb	3,+x 1,sp	f02b	18	0a	6f	6b	movb	1,-y 5,-y
ef1d	18	0a	22	01	movb	3,+x 1,x	f02f	18	0a	6f	85	movb	1,-y 5,sp
ef21	18	0a	22	41	movb	3,+x 1,y	f033	18	0d	6f	01 88	movb	1,-y ext
ef25	18	0a	22	bf	movb	3,+x 1,sp-	f038	18	0a	a8	22	movb	8,-sp 3,+x
ef29	18	0a	22	3f	movb	3,+x 1,x-	f03c	18	0a	a8	6b	movb	8,-sp 5,-y
ef2d	18	0a	22	7f	movb	3,+x 1,y-	f040	18	0a	a8	85	movb	8,-sp 5,sp
ef31	18	0a	22	8f	movb	3,+x 15,sp	f044	18	0d	a8	01 88	movb	8,-sp ext
ef35	18	0a	22	0f	movb	3,+x 15,x	f049	18	0a	28	22	movb	8,-x 3,+x
ef39	18	0a	22	4f	movb	3,+x 15,y	f04d	18	0a	28	6b	movb	8,-x 5,-y
ef3d	18	0a	22	b7	movb	3,+x 8,sp+	f051	18	0a	28	85	movb	8,-x 5,sp
ef41	18	0a	22	37	movb	3,+x 8,x+	f055	18	0d	28	01 88	movb	8,-x ext
ef45	18	0a	22	77	movb	3,+x 8,y+	f05a	18	0a	68	22	movb	8,-y 3,+x
ef49	18	0a	22	b8	movb	3,+x 8,sp-	f05e	18	0a	68	6b	movb	8,-y 5,-y
ef4d	18	0a	22	38	movb	3,+x 8,x-	f062	18	0a	68	85	movb	8,-y 5,sp
ef51	18	0a	22	78	movb	3,+x 8,y-	f066	18	0d	68	01 88	movb	8,-y ext
ef55	18	0a	22	f4	movb	3,+x a,sp	f06b	18	0a	9f	22	movb	-1,sp 3,+x
ef59	18	0a	22	e4	movb	3,+x a,x	f06f	18	0a	9f	6b	movb	-1,sp 5,-y
ef5d	18	0a	22	ec	movb	3,+x a,y	f073	18	0a	9f	85	movb	-1,sp 5,sp

f077	18	0d	9f	01	88	movb	-1,sp	ext	f190	18	0a	81	6b	movb	1,sp	5,-y	
f07c	18	0a	1f	22		movb	-1,x	3,+x	f194	18	0a	81	85	movb	1,sp	5,sp	
f080	18	0a	1f	6b		movb	-1,x	5,-y	f198	18	0d	81	01	88	movb	1,sp	ext
f084	18	0a	1f	85		movb	-1,x	5,sp	f19d	18	0a	01	22	movb	1,x	3,+x	
f088	18	0d	1f	01	88	movb	-1,x	ext	fla1	18	0a	01	6b	movb	1,x	5,-y	
f08d	18	0a	5f	22		movb	-1,y	3,+x	fla5	18	0a	01	85	movb	1,x	5,sp	
f091	18	0a	5f	6b		movb	-1,y	5,-y	fla9	18	0d	01	01	88	movb	1,x	ext
f095	18	0a	5f	85		movb	-1,y	5,sp	flae	18	0a	41	22	movb	1,y	3,+x	
f099	18	0d	5f	01	88	movb	-1,y	ext	flb2	18	0a	41	6b	movb	1,y	5,-y	
f09e	18	0a	90	22		movb	-16,sp	3,+x	flb6	18	0a	41	85	movb	1,y	5,sp	
f0a2	18	0a	90	6b		movb	-16,sp	5,-y	flba	18	0d	41	01	88	movb	1,y	ext
f0a6	18	0a	90	85		movb	-16,sp	5,sp	flbf	18	0a	bf	22	movb	1,sp-	3,+x	
f0aa	18	0d	90	01	88	movb	-16,sp	ext	flc3	18	0a	bf	6b	movb	1,sp-	5,-y	
f0af	18	0a	10	22		movb	-16,x	3,+x	flc7	18	0a	bf	85	movb	1,sp-	5,sp	
f0b3	18	0a	10	6b		movb	-16,x	5,-y	flcb	18	0d	bf	01	88	movb	1,sp-	ext
f0b7	18	0a	10	85		movb	-16,x	5,sp	fld0	18	0a	3f	22	movb	1,x-	3,+x	
f0bb	18	0d	10	01	88	movb	-16,x	ext	fld4	18	0a	3f	6b	movb	1,x-	5,-y	
f0c0	18	0a	50	22		movb	-16,y	3,+x	fld8	18	0a	3f	85	movb	1,x-	5,sp	
f0c4	18	0a	50	6b		movb	-16,y	5,-y	fldc	18	0d	3f	01	88	movb	1,x-	ext
f0c8	18	0a	50	85		movb	-16,y	5,sp	fle1	18	0a	7f	22	movb	1,y-	3,+x	
f0cc	18	0d	50	01	88	movb	-16,y	ext	fle5	18	0a	7f	6b	movb	1,y-	5,-y	
f0d1	18	0a	d2	22		movb	-small,pc	3,+x	fle9	18	0a	7f	85	movb	1,y-	5,sp	
f0d5	18	0a	d2	6b		movb	-small,pc	5,-y	fled	18	0d	7f	01	88	movb	1,y-	ext
f0d9	18	0a	d2	85		movb	-small,pc	5,sp	flf2	18	0a	6b	a0	movb	5,-y	1,+sp	
f0dd	18	0d	d2	01	88	movb	-small,pc	ext	flf6	18	0a	6b	20	movb	5,-y	1,+x	
f0e2	18	0a	92	22		movb	-small,sp	3,+x	flfa	18	0a	6b	60	movb	5,-y	1,+y	
f0e6	18	0a	92	6b		movb	-small,sp	5,-y	flfe	18	0a	6b	a7	movb	5,-y	8,+sp	
f0ea	18	0a	92	85		movb	-small,sp	5,sp	f202	18	0a	6b	27	movb	5,-y	8,+x	
f0ee	18	0d	92	01	88	movb	-small,sp	ext	f206	18	0a	6b	67	movb	5,-y	8,+y	
f0f3	18	0a	12	22		movb	-small,x	3,+x	f20a	18	0a	6b	c0	movb	5,-y	,pc	
f0f7	18	0a	12	6b		movb	-small,x	5,-y	f20e	18	0a	6b	80	movb	5,-y	,sp	
f0fb	18	0a	12	85		movb	-small,x	5,sp	f212	18	0a	6b	00	movb	5,-y	,x	
f0ff	18	0d	12	01	88	movb	-small,x	ext	f216	18	0a	6b	40	movb	5,-y	,y	
f104	18	0a	52	22		movb	-small,y	3,+x	f21a	18	0a	6b	af	movb	5,-y	1,-sp	
f108	18	0a	52	6b		movb	-small,y	5,-y	f21e	18	0a	6b	2f	movb	5,-y	1,-x	
f10c	18	0a	52	85		movb	-small,y	5,sp	f222	18	0a	6b	6f	movb	5,-y	1,-y	
f110	18	0d	52	01	88	movb	-small,y	ext	f226	18	0a	6b	a8	movb	5,-y	8,-sp	
f115	18	0a	c0	22		movb	0,pc	3,+x	f22a	18	0a	6b	28	movb	5,-y	8,-x	
f119	18	0a	c0	6b		movb	0,pc	5,-y	f22e	18	0a	6b	68	movb	5,-y	8,-y	
f11d	18	0a	c0	85		movb	0,pc	5,sp	f232	18	0a	6b	9f	movb	5,-y	-1,sp	
f121	18	0d	c0	01	88	movb	0,pc	ext	f236	18	0a	6b	1f	movb	5,-y	-1,x	
f126	18	0a	80	22		movb	0,sp	3,+x	f23a	18	0a	6b	5f	movb	5,-y	-1,y	
f12a	18	0a	80	6b		movb	0,sp	5,-y	f23e	18	0a	6b	90	movb	5,-y	-16,sp	
f12e	18	0a	80	85		movb	0,sp	5,sp	f242	18	0a	6b	10	movb	5,-y	-16,x	
f132	18	0d	80	01	88	movb	0,sp	ext	f246	18	0a	6b	50	movb	5,-y	-16,y	
f137	18	0a	00	22		movb	0,x	3,+x	f24a	18	0a	6b	d2	movb	5,-y	-small,pc	
f13b	18	0a	00	6b		movb	0,x	5,-y	f24e	18	0a	6b	92	movb	5,-y	-small,sp	
f13f	18	0a	00	85		movb	0,x	5,sp	f252	18	0a	6b	12	movb	5,-y	-small,x	
f143	18	0d	00	01	88	movb	0,x	ext	f256	18	0a	6b	52	movb	5,-y	-small,y	
f148	18	0a	40	22		movb	0,y	3,+x	f25a	18	0a	6b	c0	movb	5,-y	0,pc	
f14c	18	0a	40	6b		movb	0,y	5,-y	f25e	18	0a	6b	80	movb	5,-y	0,sp	
f150	18	0a	40	85		movb	0,y	5,sp	f262	18	0a	6b	00	movb	5,-y	0,x	
f154	18	0d	40	01	88	movb	0,y	ext	f266	18	0a	6b	40	movb	5,-y	0,y	
f159	18	0a	b0	22		movb	1,sp+	3,+x	f26a	18	0a	6b	b0	movb	5,-y	1,sp+	
f15d	18	0a	b0	6b		movb	1,sp+	5,-y	f26e	18	0a	6b	30	movb	5,-y	1,x+	
f161	18	0a	b0	85		movb	1,sp+	5,sp	f272	18	0a	6b	70	movb	5,-y	1,y+	
f165	18	0d	b0	01	88	movb	1,sp+	ext	f276	18	0a	6b	81	movb	5,-y	1,sp	
f16a	18	0a	30	22		movb	1,x+	3,+x	f27a	18	0a	6b	01	movb	5,-y	1,x	
f16e	18	0a	30	6b		movb	1,x+	5,-y	f27e	18	0a	6b	41	movb	5,-y	1,y	
f172	18	0a	30	85		movb	1,x+	5,sp	f282	18	0a	6b	bf	movb	5,-y	1,sp-	
f176	18	0d	30	01	88	movb	1,x+	ext	f286	18	0a	6b	3f	movb	5,-y	1,x-	
f17b	18	0a	70	22		movb	1,y+	3,+x	f28a	18	0a	6b	7f	movb	5,-y	1,y-	
f17f	18	0a	70	6b		movb	1,y+	5,-y	f28e	18	0a	6b	8f	movb	5,-y	15,sp	
f183	18	0a	70	85		movb	1,y+	5,sp	f292	18	0a	6b	0f	movb	5,-y	15,x	
f187	18	0d	70	01	88	movb	1,y+	ext	f296	18	0a	6b	4f	movb	5,-y	15,y	
f18c	18	0a	81	22		movb	1,sp	3,+x	f29a	18	0a	6b	b7	movb	5,-y	8,sp+	



f29e 18 0a 6b 37	movb	5,-y 8,x+	f3aa 18 0a 85 41	movb	5,sp 1,y
f2a2 18 0a 6b 77	movb	5,-y 8,y+	f3ae 18 0a 85 bf	movb	5,sp 1,sp-
f2a6 18 0a 6b b8	movb	5,-y 8,sp-	f3b2 18 0a 85 3f	movb	5,sp 1,x-
f2aa 18 0a 6b 38	movb	5,-y 8,x-	f3b6 18 0a 85 7f	movb	5,sp 1,y-
f2ae 18 0a 6b 78	movb	5,-y 8,y-	f3ba 18 0a 85 b7	movb	5,sp 8,sp+
f2b2 18 0a 6b f4	movb	5,-y a,sp	f3be 18 0a 85 37	movb	5,sp 8,x+
f2b6 18 0a 6b e4	movb	5,-y a,x	f3c2 18 0a 85 77	movb	5,sp 8,y+
f2ba 18 0a 6b ec	movb	5,-y a,y	f3c6 18 0a 85 b8	movb	5,sp 8,sp-
f2be 18 0a 6b f5	movb	5,-y b,sp	f3ca 18 0a 85 38	movb	5,sp 8,x-
f2c2 18 0a 6b e5	movb	5,-y b,x	f3ce 18 0a 85 78	movb	5,sp 8,y-
f2c6 18 0a 6b ed	movb	5,-y b,y	f3d2 18 0a 85 f4	movb	5,sp a,sp
f2ca 18 0a 6b f6	movb	5,-y d,sp	f3d6 18 0a 85 e4	movb	5,sp a,x
f2ce 18 0a 6b e6	movb	5,-y d,x	f3da 18 0a 85 ec	movb	5,sp a,y
f2d2 18 0a 6b ee	movb	5,-y d,y	f3de 18 0a 85 f5	movb	5,sp b,sp
f2d6 18 0d 6b 01 88	movb	5,-y ext	f3e2 18 0a 85 e5	movb	5,sp b,x
f2db 18 0a 6b ce	movb	5,-y small,pc	f3e6 18 0a 85 ed	movb	5,sp b,y
f2df 18 0a 6b 8e	movb	5,-y small,sp	f3ea 18 0a 85 f6	movb	5,sp d,sp
f2e3 18 0a 6b 0e	movb	5,-y small,x	f3ee 18 0a 85 e6	movb	5,sp d,x
f2e7 18 0a 6b 4e	movb	5,-y small,y	f3f2 18 0a 85 ee	movb	5,sp d,y
f2eb 18 0a 8f 22	movb	15,sp 3,+x	f3f6 18 0d 85 01 88	movb	5,sp ext
f2ef 18 0a 8f 6b	movb	15,sp 5,-y	f3fb 18 0a 85 ce	movb	5,sp small,pc
f2f3 18 0a 8f 85	movb	15,sp 5,sp	f3ff 18 0a 85 8e	movb	5,sp small,sp
f2f7 18 0d 8f 01 88	movb	15,sp ext	f403 18 0a 85 0e	movb	5,sp small,x
f2fc 18 0a 0f 22	movb	15,x 3,+x	f407 18 0a 85 4e	movb	5,sp small,y
f300 18 0a 0f 6b	movb	15,x 5,-y	f40b 18 0a b7 22	movb	8,sp+ 3,+x
f304 18 0a 0f 85	movb	15,x 5,sp	f40f 18 0a b7 6b	movb	8,sp+ 5,-y
f308 18 0d 0f 01 88	movb	15,x ext	f413 18 0a b7 85	movb	8,sp+ 5,sp
f30d 18 0a 4f 22	movb	15,y 3,+x	f417 18 0d b7 01 88	movb	8,sp+ ext
f311 18 0a 4f 6b	movb	15,y 5,-y	f41c 18 0a 37 22	movb	8,x+ 3,+x
f315 18 0a 4f 85	movb	15,y 5,sp	f420 18 0a 37 6b	movb	8,x+ 5,-y
f319 18 0d 4f 01 88	movb	15,y ext	f424 18 0a 37 85	movb	8,x+ 5,sp
f31e 18 0a 85 a0	movb	5,sp 1,+sp	f428 18 0d 37 01 88	movb	8,x+ ext
f322 18 0a 85 20	movb	5,sp 1,+x	f42d 18 0a 77 22	movb	8,y+ 3,+x
f326 18 0a 85 60	movb	5,sp 1,+y	f431 18 0a 77 6b	movb	8,y+ 5,-y
f32a 18 0a 85 a7	movb	5,sp 8,+sp	f435 18 0a 77 85	movb	8,y+ 5,sp
f32e 18 0a 85 27	movb	5,sp 8,+x	f439 18 0d 77 01 88	movb	8,y+ ext
f332 18 0a 85 67	movb	5,sp 8,+y	f43e 18 0a b8 22	movb	8,sp- 3,+x
f336 18 0a 85 c0	movb	5,sp ,pc	f442 18 0a b8 6b	movb	8,sp- 5,-y
f33a 18 0a 85 80	movb	5,sp ,sp	f446 18 0a b8 85	movb	8,sp- 5,sp
f33e 18 0a 85 00	movb	5,sp ,x	f44a 18 0d b8 01 88	movb	8,sp- ext
f342 18 0a 85 40	movb	5,sp ,y	f44f 18 0a 38 22	movb	8,x- 3,+x
f346 18 0a 85 af	movb	5,sp 1,-sp	f453 18 0a 38 6b	movb	8,x- 5,-y
f34a 18 0a 85 2f	movb	5,sp 1,-x	f457 18 0a 38 85	movb	8,x- 5,sp
f34e 18 0a 85 6f	movb	5,sp 1,-y	f45b 18 0d 38 01 88	movb	8,x- ext
f352 18 0a 85 a8	movb	5,sp 8,-sp	f460 18 0a 78 22	movb	8,y- 3,+x
f356 18 0a 85 28	movb	5,sp 8,-x	f464 18 0a 78 6b	movb	8,y- 5,-y
f35a 18 0a 85 68	movb	5,sp 8,-y	f468 18 0a 78 85	movb	8,y- 5,sp
f35e 18 0a 85 9f	movb	5,sp -1,sp	f46c 18 0d 78 01 88	movb	8,y- ext
f362 18 0a 85 1f	movb	5,sp -1,x	f471 18 0a f4 22	movb	a,sp 3,+x
f366 18 0a 85 5f	movb	5,sp -1,y	f475 18 0a f4 6b	movb	a,sp 5,-y
f36a 18 0a 85 90	movb	5,sp -16,sp	f479 18 0a f4 85	movb	a,sp 5,sp
f36e 18 0a 85 10	movb	5,sp -16,x	f47d 18 0d f4 01 88	movb	a,sp ext
f372 18 0a 85 50	movb	5,sp -16,y	f482 18 0a e4 22	movb	a,x 3,+x
f376 18 0a 85 d2	movb	5,sp -small,pc	f486 18 0a e4 6b	movb	a,x 5,-y
f37a 18 0a 85 92	movb	5,sp -small,sp	f48a 18 0a e4 85	movb	a,x 5,sp
f37e 18 0a 85 12	movb	5,sp -small,x	f48e 18 0d e4 01 88	movb	a,x ext
f382 18 0a 85 52	movb	5,sp -small,y	f493 18 0a ec 22	movb	a,y 3,+x
f386 18 0a 85 c0	movb	5,sp 0,pc	f497 18 0a ec 6b	movb	a,y 5,-y
f38a 18 0a 85 80	movb	5,sp 0,sp	f49b 18 0a ec 85	movb	a,y 5,sp
f38e 18 0a 85 00	movb	5,sp 0,x	f49f 18 0d ec 01 88	movb	a,y ext
f392 18 0a 85 40	movb	5,sp 0,y	f4a4 18 0a f5 22	movb	b,sp 3,+x
f396 18 0a 85 b0	movb	5,sp 1,sp+	f4a8 18 0a f5 6b	movb	b,sp 5,-y
f39a 18 0a 85 30	movb	5,sp 1,x+	f4ac 18 0a f5 85	movb	b,sp 5,sp
f39e 18 0a 85 70	movb	5,sp 1,y+	f4b0 18 0d f5 01 88	movb	b,sp ext
f3a2 18 0a 85 81	movb	5,sp 1,sp	f4b5 18 0a e5 22	movb	b,x 3,+x
f3a6 18 0a 85 01	movb	5,sp 1,x	f4b9 18 0a e5 6b	movb	b,x 5,-y

f4bd	18	0a	e5	85	movb	b,x	5,sp	f5fa	18	09	f5	01	88	movb	ext	b,sp		
f4c1	18	0d	e5	01	88	movb	b,x	ext	f5ff	18	09	e5	01	88	movb	ext	b,x	
f4c6	18	0a	ed	22	movb	b,y	3,+x	f604	18	09	ed	01	88	movb	ext	b,y		
f4ca	18	0a	ed	6b	movb	b,y	5,-y	f609	18	09	f6	01	88	movb	ext	d,sp		
f4ce	18	0a	ed	85	movb	b,y	5,sp	f60e	18	09	e6	01	88	movb	ext	d,x		
f4d2	18	0d	ed	01	88	movb	b,y	ext	f613	18	09	ee	01	88	movb	ext	d,y	
f4d7	18	0a	f6	22	movb	d,sp	3,+x	f618	18	0c	01	88	01	88	movb	ext	ext	
f4db	18	0a	f6	6b	movb	d,sp	5,-y	f61e	18	09	ce	01	88	movb	ext	small,pc		
f4df	18	0a	f6	85	movb	d,sp	5,sp	f623	18	09	8e	01	88	movb	ext	small,sp		
f4e3	18	0d	f6	01	88	movb	d,sp	ext	f628	18	09	0e	01	88	movb	ext	small,x	
f4e8	18	0a	e6	22	movb	d,x	3,+x	f62d	18	09	4e	01	88	movb	ext	small,y		
f4ec	18	0a	e6	6b	movb	d,x	5,-y	f632	18	0a	ce	22	movb	small,pc	3,+x			
f4f0	18	0a	e6	85	movb	d,x	5,sp	f636	18	0a	ce	6b	movb	small,pc	5,-y			
f4f4	18	0d	e6	01	88	movb	d,x	ext	f63a	18	0a	ce	85	movb	small,pc	5,sp		
f4f9	18	0a	ee	22	movb	d,y	3,+x	f63e	18	0d	ce	01	88	movb	small,pc	ext		
f4fd	18	0a	ee	6b	movb	d,y	5,-y	f643	18	0a	8e	22	movb	small,sp	3,+x			
f501	18	0a	ee	85	movb	d,y	5,sp	f647	18	0a	8e	6b	movb	small,sp	5,-y			
f505	18	0d	ee	01	88	movb	d,y	ext	f64b	18	0a	8e	85	movb	small,sp	5,sp		
f50a	18	09	a0	01	88	movb	ext	1,+sp	f64f	18	0d	8e	01	88	movb	small,sp	ext	
f50f	18	09	20	01	88	movb	ext	1,+x	f654	18	0a	0e	22	movb	small,x	3,+x		
f514	18	09	60	01	88	movb	ext	1,+y	f658	18	0a	0e	6b	movb	small,x	5,-y		
f519	18	09	a7	01	88	movb	ext	8,+sp	f65c	18	0a	0e	85	movb	small,x	5,sp		
f51e	18	09	27	01	88	movb	ext	8,+x	f660	18	0d	0e	01	88	movb	small,x	ext	
f523	18	09	67	01	88	movb	ext	8,+y	f665	18	0a	4e	22	movb	small,y	3,+x		
f528	18	09	c0	01	88	movb	ext	,pc	f669	18	0a	4e	6b	movb	small,y	5,-y		
f52d	18	09	80	01	88	movb	ext	,sp	f66d	18	0a	4e	85	movb	small,y	5,sp		
f532	18	09	00	01	88	movb	ext	,x	f671	18	0d	4e	01	88	movb	small,y	ext	
f537	18	09	40	01	88	movb	ext	,y	f676	18	05	82	01	88	movw	2,sp	ext	
f53c	18	09	af	01	88	movb	ext	1,-sp	f67b	18	02	82	0c	movw	2,sp	12,x		
f541	18	09	2f	01	88	movb	ext	1,-x	f67f	18	01	02	01	88	movw	ext	2,x	
f546	18	09	6f	01	88	movb	ext	1,-y	f684	18	02	02	00	movw	2,x	0,x		
f54b	18	09	a8	01	88	movb	ext	8,-sp	f688	18	01	ae	01	88	movw	ext	2,-sp	
f550	18	09	28	01	88	movb	ext	8,-x	f68d	18	00	ae	00	72	movw	#immed	2,-sp	
f555	18	09	68	01	88	movb	ext	8,-y	f692	18	00	ae	00	72	movw	#immed	2,-sp	
f55a	18	09	9f	01	88	movb	ext	-1,sp	f697	18	00	22	00	72	movw	#immed	3,+x	
f55f	18	09	1f	01	88	movb	ext	-1,x	f69c	18	00	6b	00	72	movw	#immed	5,-y	
f564	18	09	5f	01	88	movb	ext	-1,y	f6a1	18	00	85	00	72	movw	#immed	5,sp	
f569	18	09	90	01	88	movb	ext	-16,sp	f6a6	18	03	00	72	01	88	movw	#immed	ext
f56e	18	09	10	01	88	movb	ext	-16,x	f6ac	18	02	a0	22	movw	1,+sp	3,+x		
f573	18	09	50	01	88	movb	ext	-16,y	f6b0	18	02	a0	6b	movw	1,+sp	5,-y		
f578	18	09	d2	01	88	movb	ext	-small,pc	f6b4	18	02	a0	85	movw	1,+sp	5,sp		
f57d	18	09	92	01	88	movb	ext	-small,sp	f6b8	18	05	a0	01	88	movw	1,+sp	ext	
f582	18	09	12	01	88	movb	ext	-small,x	f6bd	18	02	20	22	movw	1,+x	3,+x		
f587	18	09	52	01	88	movb	ext	-small,y	f6c1	18	02	20	6b	movw	1,+x	5,-y		
f58c	18	09	c0	01	88	movb	ext	0,pc	f6c5	18	02	20	85	movw	1,+x	5,sp		
f591	18	09	80	01	88	movb	ext	0,sp	f6c9	18	05	20	01	88	movw	1,+x	ext	
f596	18	09	00	01	88	movb	ext	0,x	f6ce	18	02	60	22	movw	1,+y	3,+x		
f59b	18	09	40	01	88	movb	ext	0,y	f6d2	18	02	60	6b	movw	1,+y	5,-y		
f5a0	18	09	b0	01	88	movb	ext	1,sp+	f6d6	18	02	60	85	movw	1,+y	5,sp		
f5a5	18	09	30	01	88	movb	ext	1,x+	f6da	18	05	60	01	88	movw	1,+y	ext	
f5aa	18	09	70	01	88	movb	ext	1,y+	f6df	18	02	22	a0	movw	3,+x	1,+sp		
f5af	18	09	81	01	88	movb	ext	1,sp	f6e3	18	02	22	20	movw	3,+x	1,+x		
f5b4	18	09	01	01	88	movb	ext	1,x	f6e7	18	02	22	60	movw	3,+x	1,+y		
f5b9	18	09	41	01	88	movb	ext	1,y	f6eb	18	02	22	a7	movw	3,+x	8,+sp		
f5be	18	09	bf	01	88	movb	ext	1,sp-	f6ef	18	02	22	27	movw	3,+x	8,+x		
f5c3	18	09	3f	01	88	movb	ext	1,x-	f6f3	18	02	22	67	movw	3,+x	8,+y		
f5c8	18	09	7f	01	88	movb	ext	1,y-	f6f7	18	02	22	c0	movw	3,+x	,pc		
f5cd	18	09	b7	01	88	movb	ext	8,sp+	f6fb	18	02	22	80	movw	3,+x	,sp		
f5d2	18	09	37	01	88	movb	ext	8,x+	f6ff	18	02	22	00	movw	3,+x	,x		
f5d7	18	09	77	01	88	movb	ext	8,y+	f703	18	02	22	40	movw	3,+x	,y		
f5dc	18	09	b8	01	88	movb	ext	8,sp-	f707	18	02	22	af	movw	3,+x	1,-sp		
f5e1	18	09	38	01	88	movb	ext	8,x-	f70b	18	02	22	2f	movw	3,+x	1,-x		
f5e6	18	09	78	01	88	movb	ext	8,y-	f70f	18	02	22	6f	movw	3,+x	1,-y		
f5eb	18	09	f4	01	88	movb	ext	a,sp	f713	18	02	22	a8	movw	3,+x	8,-sp		
f5f0	18	09	e4	01	88	movb	ext	a,x	f717	18	02	22	28	movw	3,+x	8,-x		
f5f5	18	09	ec	01	88	movb	ext	a,y	f71b	18	02	22	68	movw	3,+x	8,-y		



f71f	18	02	22	9f	movw	3,+x -1,sp	f82d	18	01	00	00	00	movw	,x ext	
f723	18	02	22	1f	movw	3,+x -1,x	f832	18	02	40	22		movw	,y 3,+x	
f727	18	02	22	5f	movw	3,+x -1,y	f836	18	02	40	6b		movw	,y 5,-y	
f72b	18	02	22	90	movw	3,+x -16,sp	f83a	18	02	40	85		movw	,y 5,sp	
f72f	18	02	22	10	movw	3,+x -16,x	f83e	18	01	40	00	00	movw	,y ext	
f733	18	02	22	50	movw	3,+x -16,y	f843	18	02	af	22		movw	1,-sp 3,+x	
f737	18	02	22	d2	movw	3,+x -small,pc	f847	18	02	af	6b		movw	1,-sp 5,-y	
f73b	18	02	22	92	movw	3,+x -small,sp	f84b	18	02	af	85		movw	1,-sp 5,sp	
f73f	18	02	22	12	movw	3,+x -small,x	f84f	18	05	af	01	88	movw	1,-sp ext	
f743	18	02	22	52	movw	3,+x -small,y	f854	18	02	2f	22		movw	1,-x 3,+x	
f747	18	02	22	c0	movw	3,+x 0,pc	f858	18	02	2f	6b		movw	1,-x 5,-y	
f74b	18	02	22	80	movw	3,+x 0,sp	f85c	18	02	2f	85		movw	1,-x 5,sp	
f74f	18	02	22	00	movw	3,+x 0,x	f860	18	05	2f	01	88	movw	1,-x ext	
f753	18	02	22	40	movw	3,+x 0,y	f865	18	02	6f	22		movw	1,-y 3,+x	
f757	18	02	22	b0	movw	3,+x 1,sp+	f869	18	02	6f	6b		movw	1,-y 5,-y	
f75b	18	02	22	30	movw	3,+x 1,x+	f86d	18	02	6f	85		movw	1,-y 5,sp	
f75f	18	02	22	70	movw	3,+x 1,y+	f871	18	05	6f	01	88	movw	1,-y ext	
f763	18	02	22	81	movw	3,+x 1,sp	f876	18	02	a8	22		movw	8,-sp 3,+x	
f767	18	02	22	01	movw	3,+x 1,x	f87a	18	02	a8	6b		movw	8,-sp 5,-y	
f76b	18	02	22	41	movw	3,+x 1,y	f87e	18	02	a8	85		movw	8,-sp 5,sp	
f76f	18	02	22	bf	movw	3,+x 1,sp-	f882	18	05	a8	01	88	movw	8,-sp ext	
f773	18	02	22	3f	movw	3,+x 1,x-	f887	18	02	28	22		movw	8,-x 3,+x	
f777	18	02	22	7f	movw	3,+x 1,y-	f88b	18	02	28	6b		movw	8,-x 5,-y	
f77b	18	02	22	b7	movw	3,+x 8,sp+	f88f	18	02	28	85		movw	8,-x 5,sp	
f77f	18	02	22	37	movw	3,+x 8,x+	f893	18	05	28	01	88	movw	8,-x ext	
f783	18	02	22	77	movw	3,+x 8,y+	f898	18	02	68	22		movw	8,-y 3,+x	
f787	18	02	22	b8	movw	3,+x 8,sp-	f89c	18	02	68	6b		movw	8,-y 5,-y	
f78b	18	02	22	38	movw	3,+x 8,x-	f8a0	18	02	68	85		movw	8,-y 5,sp	
f78f	18	02	22	78	movw	3,+x 8,y-	f8a4	18	05	68	01	88	movw	8,-y ext	
f793	18	02	22	f4	movw	3,+x a,sp	f8a9	18	02	9f	22		movw	-1,sp 3,+x	
f797	18	02	22	e4	movw	3,+x a,x	f8ad	18	02	9f	6b		movw	-1,sp 5,-y	
f79b	18	02	22	ec	movw	3,+x a,y	f8b1	18	02	9f	85		movw	-1,sp 5,sp	
f79f	18	02	22	f5	movw	3,+x b,sp	f8b5	18	05	9f	01	88	movw	-1,sp ext	
f7a3	18	02	22	e5	movw	3,+x b,x	f8ba	18	02	1f	22		movw	-1,x 3,+x	
f7a7	18	02	22	ed	movw	3,+x b,y	f8be	18	02	1f	6b		movw	-1,x 5,-y	
f7ab	18	02	22	f6	movw	3,+x d,sp	f8c2	18	02	1f	85		movw	-1,x 5,sp	
f7af	18	02	22	e6	movw	3,+x d,x	f8c6	18	05	1f	01	88	movw	-1,x ext	
f7b3	18	02	22	ee	movw	3,+x d,y	f8cb	18	02	5f	22		movw	-1,y 3,+x	
f7b7	18	05	22	01	88	movw	3,+x ext	f8cf	18	02	5f	6b		movw	-1,y 5,-y
f7bc	18	02	22	ce	movw	3,+x small,pc	f8d3	18	02	5f	85		movw	-1,y 5,sp	
f7c0	18	02	22	8e	movw	3,+x small,sp	f8d7	18	05	5f	01	88	movw	-1,y ext	
f7c4	18	02	22	0e	movw	3,+x small,x	f8dc	18	02	90	22		movw	-16,sp 3,+x	
f7c8	18	02	22	4e	movw	3,+x small,y	f8e0	18	02	90	6b		movw	-16,sp 5,-y	
f7cc	18	02	a7	22	movw	8,+sp 3,+x	f8e4	18	02	90	85		movw	-16,sp 5,sp	
f7d0	18	02	a7	6b	movw	8,+sp 5,-y	f8e8	18	05	90	01	88	movw	-16,sp ext	
f7d4	18	02	a7	85	movw	8,+sp 5,sp	f8ed	18	02	10	22		movw	-16,x 3,+x	
f7d8	18	05	a7	01	88	movw	8,+sp ext	f8f1	18	02	10	6b		movw	-16,x 5,-y
f7dd	18	02	27	22	movw	8,+x 3,+x	f8f5	18	02	10	85		movw	-16,x 5,sp	
f7e1	18	02	27	6b	movw	8,+x 5,-y	f8f9	18	05	10	01	88	movw	-16,x ext	
f7e5	18	02	27	85	movw	8,+x 5,sp	f8fe	18	02	50	22		movw	-16,y 3,+x	
f7e9	18	05	27	01	88	movw	8,+x ext	f902	18	02	50	6b		movw	-16,y 5,-y
f7ee	18	02	67	22	movw	8,+y 3,+x	f906	18	02	50	85		movw	-16,y 5,sp	
f7f2	18	02	67	6b	movw	8,+y 5,-y	f90a	18	05	50	01	88	movw	-16,y ext	
f7f6	18	02	67	85	movw	8,+y 5,sp	f90f	18	02	d2	22		movw	-small,pc 3,+x	
f7fa	18	05	67	01	88	movw	8,+y ext	f913	18	02	d2	6b		movw	-small,pc 5,-y
f7ff	18	02	c0	22	movw	,pc 3,+x	f917	18	02	d2	85		movw	-small,pc 5,sp	
f803	18	02	c0	6b	movw	,pc 5,-y	f91b	18	05	d2	01	88	movw	-small,pc ext	
f807	18	02	c0	85	movw	,pc 5,sp	f920	18	02	92	22		movw	-small,sp 3,+x	
f80b	18	01	c0	00	00	movw	,pc ext	f924	18	02	92	6b		movw	-small,sp 5,-y
f810	18	02	80	22	movw	,sp 3,+x	f928	18	02	92	85		movw	-small,sp 5,sp	
f814	18	02	80	6b	movw	,sp 5,-y	f92c	18	05	92	01	88	movw	-small,sp ext	
f818	18	02	80	85	movw	,sp 5,sp	f931	18	02	12	22		movw	-small,x 3,+x	
f81c	18	01	80	00	00	movw	,sp ext	f935	18	02	12	6b		movw	-small,x 5,-y
f821	18	02	00	22	movw	,x 3,+x	f939	18	02	12	85		movw	-small,x 5,sp	
f825	18	02	00	6b	movw	,x 5,-y	f93d	18	05	12	01	88	movw	-small,x ext	
f829	18	02	00	85	movw	,x 5,sp	f942	18	02	52	22		movw	-small,y 3,+x	

f946	18 02 52 6b	movw	-small,y 5,-y	fa5c	18 02 6b 2f	movw	5,-y 1,-x
f94a	18 02 52 85	movw	-small,y 5,sp	fa60	18 02 6b 6f	movw	5,-y 1,-y
f94e	18 05 52 01 88	movw	-small,y ext	fa64	18 02 6b a8	movw	5,-y 8,-sp
f953	18 02 c0 22	movw	0,pc 3,+x	fa68	18 02 6b 28	movw	5,-y 8,-x
f957	18 02 c0 6b	movw	0,pc 5,-y	fa6c	18 02 6b 68	movw	5,-y 8,-y
f95b	18 02 c0 85	movw	0,pc 5,sp	fa70	18 02 6b 9f	movw	5,-y -1,sp
f95f	18 05 c0 01 88	movw	0,pc ext	fa74	18 02 6b 1f	movw	5,-y -1,x
f964	18 02 80 22	movw	0,sp 3,+x	fa78	18 02 6b 5f	movw	5,-y -1,y
f968	18 02 80 6b	movw	0,sp 5,-y	fa7c	18 02 6b 90	movw	5,-y -16,sp
f96c	18 02 80 85	movw	0,sp 5,sp	fa80	18 02 6b 10	movw	5,-y -16,x
f970	18 05 80 01 88	movw	0,sp ext	fa84	18 02 6b 50	movw	5,-y -16,y
f975	18 02 00 22	movw	0,x 3,+x	fa88	18 02 6b d2	movw	5,-y -small,pc
f979	18 02 00 6b	movw	0,x 5,-y	fa8c	18 02 6b 92	movw	5,-y -small,sp
f97d	18 02 00 85	movw	0,x 5,sp	fa90	18 02 6b 12	movw	5,-y -small,x
f981	18 05 00 01 88	movw	0,x ext	fa94	18 02 6b 52	movw	5,-y -small,y
f986	18 02 40 22	movw	0,y 3,+x	fa98	18 02 6b c0	movw	5,-y 0,pc
f98a	18 02 40 6b	movw	0,y 5,-y	fa9c	18 02 6b 80	movw	5,-y 0,sp
f98e	18 02 40 85	movw	0,y 5,sp	faa0	18 02 6b 00	movw	5,-y 0,x
f992	18 05 40 01 88	movw	0,y ext	faa4	18 02 6b 40	movw	5,-y 0,y
f997	18 02 b0 22	movw	1,sp+ 3,+x	faa8	18 02 6b b0	movw	5,-y 1,sp+
f99b	18 02 b0 6b	movw	1,sp+ 5,-y	faac	18 02 6b 30	movw	5,-y 1,x+
f99f	18 02 b0 85	movw	1,sp+ 5,sp	fab0	18 02 6b 70	movw	5,-y 1,y+
f9a3	18 05 b0 01 88	movw	1,sp+ ext	fab4	18 02 6b 81	movw	5,-y 1,sp
f9a8	18 02 30 22	movw	1,x+ 3,+x	fab8	18 02 6b 01	movw	5,-y 1,x
f9ac	18 02 30 6b	movw	1,x+ 5,-y	fabc	18 02 6b 41	movw	5,-y 1,y
f9b0	18 02 30 85	movw	1,x+ 5,sp	fac0	18 02 6b bf	movw	5,-y 1,sp-
f9b4	18 05 30 01 88	movw	1,x+ ext	fac4	18 02 6b 3f	movw	5,-y 1,x-
f9b9	18 02 70 22	movw	1,y+ 3,+x	fac8	18 02 6b 7f	movw	5,-y 1,y-
f9bd	18 02 70 6b	movw	1,y+ 5,-y	facc	18 02 6b 8f	movw	5,-y 15,sp
f9c1	18 02 70 85	movw	1,y+ 5,sp	fad0	18 02 6b 0f	movw	5,-y 15,x
f9c5	18 05 70 01 88	movw	1,y+ ext	fad4	18 02 6b 4f	movw	5,-y 15,y
f9ca	18 02 81 22	movw	1,sp 3,+x	fad8	18 02 6b b7	movw	5,-y 8,sp+
f9ce	18 02 81 6b	movw	1,sp 5,-y	fadc	18 02 6b 37	movw	5,-y 8,x+
f9d2	18 02 81 85	movw	1,sp 5,sp	fae0	18 02 6b 77	movw	5,-y 8,y+
f9d6	18 05 81 01 88	movw	1,sp ext	fae4	18 02 6b b8	movw	5,-y 8,sp-
f9db	18 02 01 22	movw	1,x 3,+x	fae8	18 02 6b 38	movw	5,-y 8,x-
f9df	18 02 01 6b	movw	1,x 5,-y	faec	18 02 6b 78	movw	5,-y 8,y-
f9e3	18 02 01 85	movw	1,x 5,sp	faf0	18 02 6b f4	movw	5,-y a,sp
f9e7	18 05 01 01 88	movw	1,x ext	faf4	18 02 6b e4	movw	5,-y a,x
f9ec	18 02 41 22	movw	1,y 3,+x	faf8	18 02 6b ec	movw	5,-y a,y
f9f0	18 02 41 6b	movw	1,y 5,-y	fafc	18 02 6b f5	movw	5,-y b,sp
f9f4	18 02 41 85	movw	1,y 5,sp	fb00	18 02 6b e5	movw	5,-y b,x
f9f8	18 05 41 01 88	movw	1,y ext	fb04	18 02 6b ed	movw	5,-y b,y
f9fd	18 02 bf 22	movw	1,sp- 3,+x	fb08	18 02 6b f6	movw	5,-y d,sp
fa01	18 02 bf 6b	movw	1,sp- 5,-y	fb0c	18 02 6b e6	movw	5,-y d,x
fa05	18 02 bf 85	movw	1,sp- 5,sp	fb10	18 02 6b ee	movw	5,-y d,y
fa09	18 05 bf 01 88	movw	1,sp- ext	fb14	18 05 6b 01 88	movw	5,-y ext
fa0e	18 02 3f 22	movw	1,x- 3,+x	fb19	18 02 6b ce	movw	5,-y small,pc
fa12	18 02 3f 6b	movw	1,x- 5,-y	fb1d	18 02 6b 8e	movw	5,-y small,sp
fa16	18 02 3f 85	movw	1,x- 5,sp	fb21	18 02 6b 0e	movw	5,-y small,x
fa1a	18 05 3f 01 88	movw	1,x- ext	fb25	18 02 6b 4e	movw	5,-y small,y
fa1f	18 02 7f 22	movw	1,y- 3,+x	fb29	18 02 8f 22	movw	15,sp 3,+x
fa23	18 02 7f 6b	movw	1,y- 5,-y	fb2d	18 02 8f 6b	movw	15,sp 5,-y
fa27	18 02 7f 85	movw	1,y- 5,sp	fb31	18 02 8f 85	movw	15,sp 5,sp
fa2b	18 05 7f 01 88	movw	1,y- ext	fb35	18 05 8f 01 88	movw	15,sp ext
fa30	18 02 6b a0	movw	5,-y 1,+sp	fb3a	18 02 0f 22	movw	15,x 3,+x
fa34	18 02 6b 20	movw	5,-y 1,+x	fb3e	18 02 0f 6b	movw	15,x 5,-y
fa38	18 02 6b 60	movw	5,-y 1,+y	fb42	18 02 0f 85	movw	15,x 5,sp
fa3c	18 02 6b a7	movw	5,-y 8,+sp	fb46	18 05 0f 01 88	movw	15,x ext
fa40	18 02 6b 27	movw	5,-y 8,+x	fb4b	18 02 4f 22	movw	15,y 3,+x
fa44	18 02 6b 67	movw	5,-y 8,+y	fb4f	18 02 4f 6b	movw	15,y 5,-y
fa48	18 02 6b c0	movw	5,-y ,pc	fb53	18 02 4f 85	movw	15,y 5,sp
fa4c	18 02 6b 80	movw	5,-y ,sp	fb57	18 05 4f 01 88	movw	15,y ext
fa50	18 02 6b 00	movw	5,-y ,x	fb5c	18 02 85 a0	movw	5,sp 1,+sp
fa54	18 02 6b 40	movw	5,-y ,y	fb60	18 02 85 20	movw	5,sp 1,+x
fa58	18 02 6b af	movw	5,-y 1,-sp	fb64	18 02 85 60	movw	5,sp 1,+y

fb68	18 02 85 a7	movw	5,sp 8,+sp	fc73	18 02 77 85	movw	8,y+ 5,sp
fb6c	18 02 85 27	movw	5,sp 8,+x	fc77	18 05 77 01 88	movw	8,y+ ext
fb70	18 02 85 67	movw	5,sp 8,+y	fc7c	18 02 b8 22	movw	8,sp- 3,+x
fb74	18 02 85 c0	movw	5,sp ,pc	fc80	18 02 b8 6b	movw	8,sp- 5,-y
fb78	18 02 85 80	movw	5,sp ,sp	fc84	18 02 b8 85	movw	8,sp- 5,sp
fb7c	18 02 85 00	movw	5,sp ,x	fc88	18 05 b8 01 88	movw	8,sp- ext
fb80	18 02 85 40	movw	5,sp ,y	fc8d	18 02 38 22	movw	8,x- 3,+x
fb84	18 02 85 af	movw	5,sp 1,-sp	fc91	18 02 38 6b	movw	8,x- 5,-y
fb88	18 02 85 2f	movw	5,sp 1,-x	fc95	18 02 38 85	movw	8,x- 5,sp
fb8c	18 02 85 6f	movw	5,sp 1,-y	fc99	18 05 38 01 88	movw	8,x- ext
fb90	18 02 85 a8	movw	5,sp 8,-sp	fc9e	18 02 78 22	movw	8,y- 3,+x
fb94	18 02 85 28	movw	5,sp 8,-x	fca2	18 02 78 6b	movw	8,y- 5,-y
fb98	18 02 85 68	movw	5,sp 8,-y	fca6	18 02 78 85	movw	8,y- 5,sp
fb9c	18 02 85 9f	movw	5,sp -1,sp	fcaa	18 05 78 01 88	movw	8,y- ext
fba0	18 02 85 1f	movw	5,sp -1,x	fcaf	18 02 f4 22	movw	a,sp 3,+x
fba4	18 02 85 5f	movw	5,sp -1,y	fc3b	18 02 f4 6b	movw	a,sp 5,-y
fba8	18 02 85 90	movw	5,sp -16,sp	fc37	18 02 f4 85	movw	a,sp 5,sp
fbac	18 02 85 10	movw	5,sp -16,x	fcbb	18 05 f4 01 88	movw	a,sp ext
fbb0	18 02 85 50	movw	5,sp -16,y	fcc0	18 02 e4 22	movw	a,x 3,+x
fb34	18 02 85 d2	movw	5,sp -small,pc	fcc4	18 02 e4 6b	movw	a,x 5,-y
fb38	18 02 85 92	movw	5,sp -small,sp	fcc8	18 02 e4 85	movw	a,x 5,sp
fb3c	18 02 85 12	movw	5,sp -small,x	fccc	18 05 e4 01 88	movw	a,x ext
fb40	18 02 85 52	movw	5,sp -small,y	fed1	18 02 ec 22	movw	a,y 3,+x
fb44	18 02 85 c0	movw	5,sp 0,pc	fed5	18 02 ec 6b	movw	a,y 5,-y
fb48	18 02 85 80	movw	5,sp 0,sp	fed9	18 02 ec 85	movw	a,y 5,sp
fb4c	18 02 85 00	movw	5,sp 0,x	fedd	18 05 ec 01 88	movw	a,y ext
fb50	18 02 85 40	movw	5,sp 0,y	fce2	18 02 f5 22	movw	b,sp 3,+x
fb54	18 02 85 b0	movw	5,sp 1,sp+	fce6	18 02 f5 6b	movw	b,sp 5,-y
fb58	18 02 85 30	movw	5,sp 1,x+	fcea	18 02 f5 85	movw	b,sp 5,sp
fb5c	18 02 85 70	movw	5,sp 1,y+	fcee	18 05 f5 01 88	movw	b,sp ext
fb60	18 02 85 81	movw	5,sp 1,sp	fcf3	18 02 e5 22	movw	b,x 3,+x
fb64	18 02 85 01	movw	5,sp 1,x	fcf7	18 02 e5 6b	movw	b,x 5,-y
fb68	18 02 85 41	movw	5,sp 1,y	fcfb	18 02 e5 85	movw	b,x 5,sp
fb6c	18 02 85 bf	movw	5,sp 1,sp-	fcff	18 05 e5 01 88	movw	b,x ext
fb70	18 02 85 3f	movw	5,sp 1,x-	fd04	18 02 ed 22	movw	b,y 3,+x
fb74	18 02 85 7f	movw	5,sp 1,y-	fd08	18 02 ed 6b	movw	b,y 5,-y
fb78	18 02 85 b7	movw	5,sp 8,sp+	fd0c	18 02 ed 85	movw	b,y 5,sp
fb7c	18 02 85 37	movw	5,sp 8,x+	fd10	18 05 ed 01 88	movw	b,y ext
fc00	18 02 85 77	movw	5,sp 8,y+	fd15	18 02 f6 22	movw	d,sp 3,+x
fc04	18 02 85 b8	movw	5,sp 8,sp-	fd19	18 02 f6 6b	movw	d,sp 5,-y
fc08	18 02 85 38	movw	5,sp 8,x-	fd1d	18 02 f6 85	movw	d,sp 5,sp
fc0c	18 02 85 78	movw	5,sp 8,y-	fd21	18 05 f6 01 88	movw	d,sp ext
fc10	18 02 85 f4	movw	5,sp a,sp	fd26	18 02 e6 22	movw	d,x 3,+x
fc14	18 02 85 e4	movw	5,sp a,x	fd2a	18 02 e6 6b	movw	d,x 5,-y
fc18	18 02 85 ec	movw	5,sp a,y	fd2e	18 02 e6 85	movw	d,x 5,sp
fc1c	18 02 85 f5	movw	5,sp b,sp	fd32	18 05 e6 01 88	movw	d,x ext
fc20	18 02 85 e5	movw	5,sp b,x	fd37	18 02 ee 22	movw	d,y 3,+x
fc24	18 02 85 ed	movw	5,sp b,y	fd3b	18 02 ee 6b	movw	d,y 5,-y
fc28	18 02 85 f6	movw	5,sp d,sp	fd3f	18 02 ee 85	movw	d,y 5,sp
fc2c	18 02 85 e6	movw	5,sp d,x	fd43	18 05 ee 01 88	movw	d,y ext
fc30	18 02 85 ee	movw	5,sp d,y	fd48	18 01 a0 01 88	movw	ext 1,+sp
fc34	18 05 85 01 88	movw	5,sp ext	fd4d	18 01 20 01 88	movw	ext 1,+x
fc39	18 02 85 ce	movw	5,sp small,pc	fd52	18 01 60 01 88	movw	ext 1,+y
fc3d	18 02 85 8e	movw	5,sp small,sp	fd57	18 01 a7 01 88	movw	ext 8,+sp
fc41	18 02 85 0e	movw	5,sp small,x	fd5c	18 01 27 01 88	movw	ext 8,+x
fc45	18 02 85 4e	movw	5,sp small,y	fd61	18 01 67 01 88	movw	ext 8,+y
fc49	18 02 b7 22	movw	8,sp+ 3,+x	fd66	18 01 c0 01 88	movw	ext ,pc
fc4d	18 02 b7 6b	movw	8,sp+ 5,-y	fd6b	18 01 80 01 88	movw	ext ,sp
fc51	18 02 b7 85	movw	8,sp+ 5,sp	fd70	18 01 00 01 88	movw	ext ,x
fc55	18 05 b7 01 88	movw	8,sp+ ext	fd75	18 01 40 01 88	movw	ext ,y
fc5a	18 02 37 22	movw	8,x+ 3,+x	fd7a	18 01 af 01 88	movw	ext 1,-sp
fc5e	18 02 37 6b	movw	8,x+ 5,-y	fd7f	18 01 2f 01 88	movw	ext 1,-x
fc62	18 02 37 85	movw	8,x+ 5,sp	fd84	18 01 6f 01 88	movw	ext 1,-y
fc66	18 05 37 01 88	movw	8,x+ ext	fd89	18 01 a8 01 88	movw	ext 8,-sp
fc6b	18 02 77 22	movw	8,y+ 3,+x	fd8e	18 01 28 01 88	movw	ext 8,-x
fc6f	18 02 77 6b	movw	8,y+ 5,-y	fd93	18 01 68 01 88	movw	ext 8,-y

fd98	18 01 9f 01 88	movw	ext -1,sp	fec1	60 c0	neg	,pc
fd9d	18 01 1f 01 88	movw	ext -1,x	fec3	60 80	neg	,sp
fda2	18 01 5f 01 88	movw	ext -1,y	fec5	60 00	neg	,x
fda7	18 01 90 01 88	movw	ext -16,sp	fec7	60 40	neg	,y
fdac	18 01 10 01 88	movw	ext -16,x	fec9	60 af	neg	1,-sp
fdb1	18 01 50 01 88	movw	ext -16,y	fecb	60 2f	neg	1,-x
fdb6	18 01 d2 01 88	movw	ext -small,pc	fecd	60 6f	neg	1,-y
fdbb	18 01 92 01 88	movw	ext -small,sp	fecf	60 a8	neg	8,-sp
fdc0	18 01 12 01 88	movw	ext -small,x	fed1	60 28	neg	8,-x
fdc5	18 01 52 01 88	movw	ext -small,y	fed3	60 68	neg	8,-y
fdca	18 01 c0 01 88	movw	ext 0,pc	fed5	60 9f	neg	-1,sp
fdcf	18 01 80 01 88	movw	ext 0,sp	fed7	60 1f	neg	-1,x
ddd4	18 01 00 01 88	movw	ext 0,x	fed9	60 5f	neg	-1,y
ddd9	18 01 40 01 88	movw	ext 0,y	fedb	60 90	neg	-16,sp
ddde	18 01 b0 01 88	movw	ext 1,sp+	fedd	60 10	neg	-16,x
dde3	18 01 30 01 88	movw	ext 1,x+	fedf	60 50	neg	-16,y
dde8	18 01 70 01 88	movw	ext 1,y+	feel	60 f1 ef	neg	-17,sp
ddeb	18 01 81 01 88	movw	ext 1,sp	fee4	60 e1 ef	neg	-17,x
fdf2	18 01 01 01 88	movw	ext 1,x	fee7	60 e9 ef	neg	-17,y
fdf7	18 01 41 01 88	movw	ext 1,y	feea	60 d2	neg	-small,pc
fdfc	18 01 bf 01 88	movw	ext 1,sp-	feec	60 92	neg	-small,sp
fe01	18 01 3f 01 88	movw	ext 1,x-	feee	60 12	neg	-small,x
fe06	18 01 7f 01 88	movw	ext 1,y-	fef0	60 52	neg	-small,y
fe0b	18 01 b7 01 88	movw	ext 8,sp+	fef2	60 c0	neg	0,pc
fe10	18 01 37 01 88	movw	ext 8,x+	fef4	60 80	neg	0,sp
fe15	18 01 77 01 88	movw	ext 8,y+	fef6	60 00	neg	0,x
fe1a	18 01 b8 01 88	movw	ext 8,sp-	fef8	60 40	neg	0,y
fe1f	18 01 38 01 88	movw	ext 8,x-	fefa	60 b0	neg	1,sp+
fe24	18 01 78 01 88	movw	ext 8,y-	fefc	60 30	neg	1,x+
fe29	18 01 f4 01 88	movw	ext a,sp	fefe	60 70	neg	1,y+
fe2e	18 01 e4 01 88	movw	ext a,x	ff00	60 81	neg	1,sp
fe33	18 01 ec 01 88	movw	ext a,y	ff02	60 01	neg	1,x
fe38	18 01 f5 01 88	movw	ext b,sp	ff04	60 41	neg	1,y
fe3d	18 01 e5 01 88	movw	ext b,x	ff06	60 bf	neg	1,sp-
fe42	18 01 ed 01 88	movw	ext b,y	ff08	60 3f	neg	1,x-
fe47	18 01 f6 01 88	movw	ext d,sp	ff0a	60 7f	neg	1,y-
fe4c	18 01 e6 01 88	movw	ext d,x	ff0c	60 f8 7d	neg	125,pc
fe51	18 01 ee 01 88	movw	ext d,y	ff0f	60 f0 7d	neg	125,sp
fe56	18 04 01 88 01 88	movw	ext ext	ff12	60 e0 7d	neg	125,x
fe5c	18 01 ce 01 88	movw	ext small,pc	ff15	60 e8 7d	neg	125,y
fe61	18 01 8e 01 88	movw	ext small,sp	ff18	60 8f	neg	15,sp
fe66	18 01 0e 01 88	movw	ext small,x	ff1a	60 0f	neg	15,x
fe6b	18 01 4e 01 88	movw	ext small,y	ff1c	60 4f	neg	15,y
fe70	18 02 ce 22	movw	small,pc 3,+x	ff1e	60 f0 10	neg	16,sp
fe74	18 02 ce 6b	movw	small,pc 5,-y	ff21	60 e0 10	neg	16,x
fe78	18 02 ce 85	movw	small,pc 5,sp	ff24	60 e8 10	neg	16,y
fe7c	18 05 ce 01 88	movw	small,pc ext	ff27	60 b7	neg	8,sp+
fe81	18 02 8e 22	movw	small,sp 3,+x	ff29	60 37	neg	8,x+
fe85	18 02 8e 6b	movw	small,sp 5,-y	ff2b	60 77	neg	8,y+
fe89	18 02 8e 85	movw	small,sp 5,sp	ff2d	60 b8	neg	8,sp-
fe8d	18 05 8e 01 88	movw	small,sp ext	ff2f	60 38	neg	8,x-
fe92	18 02 0e 22	movw	small,x 3,+x	ff31	60 78	neg	8,y-
fe96	18 02 0e 6b	movw	small,x 5,-y	ff33	60 f4	neg	a,sp
fe9a	18 02 0e 85	movw	small,x 5,sp	ff35	60 e4	neg	a,x
fe9e	18 05 0e 01 88	movw	small,x ext	ff37	60 ec	neg	a,y
fea3	18 02 4e 22	movw	small,y 3,+x	ff39	60 f5	neg	b,sp
fea7	18 02 4e 6b	movw	small,y 5,-y	ff3b	60 e5	neg	b,x
feab	18 02 4e 85	movw	small,y 5,sp	ff3d	60 ed	neg	b,y
feaf	18 05 4e 01 88	movw	small,y ext	ff3f	60 f6	neg	d,sp
feb4	12	mul		ff41	60 e6	neg	d,x
feb5	60 a0	neg	1,+sp	ff43	60 ee	neg	d,y
feb7	60 20	neg	1,+x	ff45	70 00 55	neg	dir
feb9	60 60	neg	1,+y	ff48	70 01 88	neg	ext
febb	60 a7	neg	8,+sp	ff4b	70 01 88	neg	ext
febd	60 27	neg	8,+x	ff4e	60 f2 01 88	neg	ext,sp
febf	60 67	neg	8,+y	ff52	60 e2 01 88	neg	ext,x

ff56 60 ea 01 88	neg	ext,y	ffe7 aa b7	oraa	8,sp+
ff5a 60 f8 37	neg	ind,pc	ffe9 aa 37	oraa	8,x+
ff5d 60 f0 37	neg	ind,sp	ffeb aa 77	oraa	8,y+
ff60 60 e0 37	neg	ind,x	ffed aa b8	oraa	8,sp-
ff63 60 e8 37	neg	ind,y	ffef aa 38	oraa	8,x-
ff66 60 ce	neg	small,pc	fff1 aa 78	oraa	8,y-
ff68 60 8e	neg	small,sp	fff3 aa f4	oraa	a,sp
ff6a 60 0e	neg	small,x	fff5 aa e4	oraa	a,x
ff6c 60 4e	neg	small,y	fff7 aa ec	oraa	a,y
ff6e 40	nega		fff9 aa f5	oraa	b,sp
ff6f 50	negb		fffb aa e5	oraa	b,x
ff70 a7	nop		fffd aa ed	oraa	b,y
ff71 8a 72	oraa	#immed	ffff aa f6	oraa	d,sp
ff73 8a 72	oraa	#immed	0001 aa e6	oraa	d,x
ff75 aa a0	oraa	1,+sp	0003 aa ee	oraa	d,y
ff77 aa 20	oraa	1,+x	0005 9a 55	oraa	dir
ff79 aa 60	oraa	1,+y	0007 9a 55	oraa	dir
ff7b aa a7	oraa	8,+sp	0009 ba 01 88	oraa	ext
ff7d aa 27	oraa	8,+x	000c ba 01 88	oraa	ext
ff7f aa 67	oraa	8,+y	000f aa f2 01 88	oraa	ext,sp
ff81 aa c0	oraa	,pc	0013 aa e2 01 88	oraa	ext,x
ff83 aa 80	oraa	,sp	0017 aa ea 01 88	oraa	ext,y
ff85 aa 00	oraa	,x	001b aa f8 37	oraa	ind,pc
ff87 aa 40	oraa	,y	001e aa f0 37	oraa	ind,sp
ff89 aa af	oraa	1,-sp	0021 aa e0 37	oraa	ind,x
ff8b aa 2f	oraa	1,-x	0024 aa e8 37	oraa	ind,y
ff8d aa 6f	oraa	1,-y	0027 aa ce	oraa	small,pc
ff8f aa a8	oraa	8,-sp	0029 aa 8e	oraa	small,sp
ff91 aa 28	oraa	8,-x	002b aa 0e	oraa	small,x
ff93 aa 68	oraa	8,-y	002d aa 4e	oraa	small,y
ff95 aa 9f	oraa	-1,sp	002f ca 72	orab	#immed
ff97 aa 1f	oraa	-1,x	0031 ca 72	orab	#immed
ff99 aa 5f	oraa	-1,y	0033 ea a0	orab	1,+sp
ff9b aa 90	oraa	-16,sp	0035 ea 20	orab	1,+x
ff9d aa 10	oraa	-16,x	0037 ea 60	orab	1,+y
ff9f aa 50	oraa	-16,y	0039 ea a7	orab	8,+sp
ffa1 aa f1 ef	oraa	-17,sp	003b ea 27	orab	8,+x
ffa4 aa e1 ef	oraa	-17,x	003d ea 67	orab	8,+y
ffa7 aa e9 ef	oraa	-17,y	003f ea c0	orab	,pc
ffaa aa d2	oraa	-small,pc	0041 ea 80	orab	,sp
ffac aa 92	oraa	-small,sp	0043 ea 00	orab	,x
ffae aa 12	oraa	-small,x	0045 ea 40	orab	,y
ffb0 aa 52	oraa	-small,y	0047 ea af	orab	1,-sp
ffb2 aa c0	oraa	0,pc	0049 ea 2f	orab	1,-x
ffb4 aa 80	oraa	0,sp	004b ea 6f	orab	1,-y
ffb6 aa 00	oraa	0,x	004d ea a8	orab	8,-sp
ffb8 aa 40	oraa	0,y	004f ea 28	orab	8,-x
ffba aa b0	oraa	1,sp+	0051 ea 68	orab	8,-y
ffbc aa 30	oraa	1,x+	0053 ea 9f	orab	-1,sp
ffbe aa 70	oraa	1,y+	0055 ea 1f	orab	-1,x
ffc0 aa 81	oraa	1,sp	0057 ea 5f	orab	-1,y
ffc2 aa 01	oraa	1,x	0059 ea 90	orab	-16,sp
ffc4 aa 41	oraa	1,y	005b ea 10	orab	-16,x
ffc6 aa bf	oraa	1,sp-	005d ea 50	orab	-16,y
ffc8 aa 3f	oraa	1,x-	005f ea f1 ef	orab	-17,sp
ffca aa 7f	oraa	1,y-	0062 ea e1 ef	orab	-17,x
ffcc aa f8 7d	oraa	125,pc	0065 ea e9 ef	orab	-17,y
ffcf aa f0 7d	oraa	125,sp	0068 ea d2	orab	-small,pc
ffd2 aa e0 7d	oraa	125,x	006a ea 92	orab	-small,sp
ffd5 aa e8 7d	oraa	125,y	006c ea 12	orab	-small,x
ffd8 aa 8f	oraa	15,sp	006e ea 52	orab	-small,y
ffda aa 0f	oraa	15,x	0070 ea c0	orab	0,pc
ffdc aa 4f	oraa	15,y	0072 ea 80	orab	0,sp
ffde aa f0 10	oraa	16,sp	0074 ea 00	orab	0,x
ffe1 aa e0 10	oraa	16,x	0076 ea 40	orab	0,y
ffe4 aa e8 10	oraa	16,y	0078 ea b0	orab	1,sp+

007a	ea 30	orab	1,x+	0106	65 67	rol	8,+y
007c	ea 70	orab	1,y+	0108	65 c0	rol	,pc
007e	ea 81	orab	1,sp	010a	65 80	rol	,sp
0080	ea 01	orab	1,x	010c	65 00	rol	,x
0082	ea 41	orab	1,y	010e	65 40	rol	,y
0084	ea bf	orab	1,sp-	0110	65 af	rol	1,-sp
0086	ea 3f	orab	1,x-	0112	65 2f	rol	1,-x
0088	ea 7f	orab	1,y-	0114	65 6f	rol	1,-y
008a	ea f8 7d	orab	125,pc	0116	65 a8	rol	8,-sp
008d	ea f0 7d	orab	125,sp	0118	65 28	rol	8,-x
0090	ea e0 7d	orab	125,x	011a	65 68	rol	8,-y
0093	ea e8 7d	orab	125,y	011c	65 9f	rol	-1,sp
0096	ea 8f	orab	15,sp	011e	65 1f	rol	-1,x
0098	ea 0f	orab	15,x	0120	65 5f	rol	-1,y
009a	ea 4f	orab	15,y	0122	65 90	rol	-16,sp
009c	ea f0 10	orab	16,sp	0124	65 10	rol	-16,x
009f	ea e0 10	orab	16,x	0126	65 50	rol	-16,y
00a2	ea e8 10	orab	16,y	0128	65 f1 ef	rol	-17,sp
00a5	ea b7	orab	8,sp+	012b	65 e1 ef	rol	-17,x
00a7	ea 37	orab	8,x+	012e	65 e9 ef	rol	-17,y
00a9	ea 77	orab	8,y+	0131	65 d2	rol	-small,pc
00ab	ea b8	orab	8,sp-	0133	65 92	rol	-small,sp
00ad	ea 38	orab	8,x-	0135	65 12	rol	-small,x
00af	ea 78	orab	8,y-	0137	65 52	rol	-small,y
00b1	ea f4	orab	a,sp	0139	65 c0	rol	0,pc
00b3	ea e4	orab	a,x	013b	65 80	rol	0,sp
00b5	ea ec	orab	a,y	013d	65 00	rol	0,x
00b7	ea f5	orab	b,sp	013f	65 40	rol	0,y
00b9	ea e5	orab	b,x	0141	65 b0	rol	1,sp+
00bb	ea ed	orab	b,y	0143	65 30	rol	1,x+
00bd	ea f6	orab	d,sp	0145	65 70	rol	1,y+
00bf	ea e6	orab	d,x	0147	65 81	rol	1,sp
00c1	ea ee	orab	d,y	0149	65 01	rol	1,x
00c3	da 55	orab	dir	014b	65 41	rol	1,y
00c5	da 55	orab	dir	014d	65 bf	rol	1,sp-
00c7	fa 01 88	orab	ext	014f	65 3f	rol	1,x-
00ca	fa 01 88	orab	ext	0151	65 7f	rol	1,y-
00cd	ea f2 01 88	orab	ext,sp	0153	65 f8 7d	rol	125,pc
00d1	ea e2 01 88	orab	ext,x	0156	65 f0 7d	rol	125,sp
00d5	ea ea 01 88	orab	ext,y	0159	65 e0 7d	rol	125,x
00d9	ea f8 37	orab	ind,pc	015c	65 e8 7d	rol	125,y
00dc	ea f0 37	orab	ind,sp	015f	65 8f	rol	15,sp
00df	ea e0 37	orab	ind,x	0161	65 0f	rol	15,x
00e2	ea e8 37	orab	ind,y	0163	65 4f	rol	15,y
00e5	ea ce	orab	small,pc	0165	65 f0 10	rol	16,sp
00e7	ea 8e	orab	small,sp	0168	65 e0 10	rol	16,x
00e9	ea 0e	orab	small,x	016b	65 e8 10	rol	16,y
00eb	ea 4e	orab	small,y	016e	65 b7	rol	8,sp+
00ed	14 72	orcc	#immed	0170	65 37	rol	8,x+
00ef	36	psha		0172	65 77	rol	8,y+
00f0	37	pshb		0174	65 b8	rol	8,sp-
00f1	3b	pshd		0176	65 38	rol	8,x-
00f2	34	pshx		0178	65 78	rol	8,y-
00f3	35	pshy		017a	65 f4	rol	a,sp
00f4	32	pula		017c	65 e4	rol	a,x
00f5	33	pulb		017e	65 ec	rol	a,y
00f6	38	pulc		0180	65 f5	rol	b,sp
00f7	3a	puld		0182	65 e5	rol	b,x
00f8	30	pulx		0184	65 ed	rol	b,y
00f9	31	puly		0186	65 f6	rol	d,sp
00fa	18 3a	rev		0188	65 e6	rol	d,x
00fc	65 a0	rol	1,+sp	018a	65 ee	rol	d,y
00fe	65 20	rol	1,+x	018c	75 00 55	rol	dir
0100	65 60	rol	1,+y	018f	75 01 88	rol	ext
0102	65 a7	rol	8,+sp	0192	75 01 88	rol	ext
0104	65 27	rol	8,+x	0195	65 f2 01 88	rol	ext,sp



0199 65 e2 01 88	rol	ext,x	022d 66 77	ror	8,y+
019d 65 ea 01 88	rol	ext,y	022f 66 b8	ror	8,sp-
01a1 65 f8 37	rol	ind,pc	0231 66 38	ror	8,x-
01a4 65 f0 37	rol	ind,sp	0233 66 78	ror	8,y-
01a7 65 e0 37	rol	ind,x	0235 66 f4	ror	a,sp
01aa 65 e8 37	rol	ind,y	0237 66 e4	ror	a,x
01ad 65 ce	rol	small,pc	0239 66 ec	ror	a,y
01af 65 8e	rol	small,sp	023b 66 f5	ror	b,sp
01b1 65 0e	rol	small,x	023d 66 e5	ror	b,x
01b3 65 4e	rol	small,y	023f 66 ed	ror	b,y
01b5 45	rola		0241 66 f6	ror	d,sp
01b6 55	rolb		0243 66 e6	ror	d,x
01b7 66 a0	ror	1,+sp	0245 66 ee	ror	d,y
01b9 66 20	ror	1,+x	0247 76 00 55	ror	dir
01bb 66 60	ror	1,y	024a 76 01 88	ror	ext
01bd 66 a7	ror	8,+sp	024d 76 01 88	ror	ext
01bf 66 27	ror	8,+x	0250 66 f2 01 88	ror	ext,sp
01c1 66 67	ror	8,+y	0254 66 e2 01 88	ror	ext,x
01c3 66 c0	ror	,pc	0258 66 ea 01 88	ror	ext,y
01c5 66 80	ror	,sp	025c 66 f8 37	ror	ind,pc
01c7 66 00	ror	,x	025f 66 f0 37	ror	ind,sp
01c9 66 40	ror	,y	0262 66 e0 37	ror	ind,x
01cb 66 af	ror	1,-sp	0265 66 e8 37	ror	ind,y
01cd 66 2f	ror	1,-x	0268 66 ce	ror	small,pc
01cf 66 6f	ror	1,-y	026a 66 8e	ror	small,sp
01d1 66 a8	ror	8,-sp	026c 66 0e	ror	small,x
01d3 66 28	ror	8,-x	026e 66 4e	ror	small,y
01d5 66 68	ror	8,-y	0270 46	rora	
01d7 66 9f	ror	-1,sp	0271 56	rorb	
01d9 66 1f	ror	-1,x	0272 0b	rti	
01db 66 5f	ror	-1,y	0273 3d	rts	
01dd 66 90	ror	-16,sp	0274 18 16	sba	
01df 66 10	ror	-16,x	0276 82 72	sbca	#immed
01e1 66 50	ror	-16,y	0278 82 72	sbca	#immed
01e3 66 f1 ef	ror	-17,sp	027a a2 a0	sbca	1,+sp
01e6 66 e1 ef	ror	-17,x	027c a2 20	sbca	1,+x
01e9 66 e9 ef	ror	-17,y	027e a2 60	sbca	1,+y
01ec 66 d2	ror	-small,pc	0280 a2 a7	sbca	8,+sp
01ee 66 92	ror	-small,sp	0282 a2 27	sbca	8,+x
01f0 66 12	ror	-small,x	0284 a2 67	sbca	8,+y
01f2 66 52	ror	-small,y	0286 a2 c0	sbca	,pc
01f4 66 c0	ror	0,pc	0288 a2 80	sbca	,sp
01f6 66 80	ror	0,sp	028a a2 00	sbca	,x
01f8 66 00	ror	0,x	028c a2 40	sbca	,y
01fa 66 40	ror	0,y	028e a2 af	sbca	1,-sp
01fc 66 b0	ror	1,sp+	0290 a2 2f	sbca	1,-x
01fe 66 30	ror	1,x+	0292 a2 6f	sbca	1,-y
0200 66 70	ror	1,y+	0294 a2 a8	sbca	8,-sp
0202 66 81	ror	1,sp	0296 a2 28	sbca	8,-x
0204 66 01	ror	1,x	0298 a2 68	sbca	8,-y
0206 66 41	ror	1,y	029a a2 9f	sbca	-1,sp
0208 66 bf	ror	1,sp-	029c a2 1f	sbca	-1,x
020a 66 3f	ror	1,x-	029e a2 5f	sbca	-1,y
020c 66 7f	ror	1,y-	02a0 a2 90	sbca	-16,sp
020e 66 f8 7d	ror	125,pc	02a2 a2 10	sbca	-16,x
0211 66 f0 7d	ror	125,sp	02a4 a2 50	sbca	-16,y
0214 66 e0 7d	ror	125,x	02a6 a2 f1 ef	sbca	-17,sp
0217 66 e8 7d	ror	125,y	02a9 a2 e1 ef	sbca	-17,x
021a 66 8f	ror	15,sp	02ac a2 e9 ef	sbca	-17,y
021c 66 0f	ror	15,x	02af a2 d2	sbca	-small,pc
021e 66 4f	ror	15,y	02b1 a2 92	sbca	-small,sp
0220 66 f0 10	ror	16,sp	02b3 a2 12	sbca	-small,x
0223 66 e0 10	ror	16,x	02b5 a2 52	sbca	-small,y
0226 66 e8 10	ror	16,y	02b7 a2 c0	sbca	0,pc
0229 66 b7	ror	8,sp+	02b9 a2 80	sbca	0,sp
022b 66 37	ror	8,x+	02bb a2 00	sbca	0,x

02bd	a2	40	sbca	0,y	0354	e2	28	sbc	8,-x
02bf	a2	b0	sbca	1,sp+	0356	e2	68	sbc	8,-y
02c1	a2	30	sbca	1,x+	0358	e2	9f	sbc	-1,sp
02c3	a2	70	sbca	1,y+	035a	e2	1f	sbc	-1,x
02c5	a2	81	sbca	1,sp	035c	e2	5f	sbc	-1,y
02c7	a2	01	sbca	1,x	035e	e2	90	sbc	-16,sp
02c9	a2	41	sbca	1,y	0360	e2	10	sbc	-16,x
02cb	a2	bf	sbca	1,sp-	0362	e2	50	sbc	-16,y
02cd	a2	3f	sbca	1,x-	0364	e2	f1 ef	sbc	-17,sp
02cf	a2	7f	sbca	1,y-	0367	e2	e1 ef	sbc	-17,x
02d1	a2	f8 7d	sbca	125,pc	036a	e2	e9 ef	sbc	-17,y
02d4	a2	f0 7d	sbca	125,sp	036d	e2	d2	sbc	-small,pc
02d7	a2	e0 7d	sbca	125,x	036f	e2	92	sbc	-small,sp
02da	a2	e8 7d	sbca	125,y	0371	e2	12	sbc	-small,x
02dd	a2	8f	sbca	15,sp	0373	e2	52	sbc	-small,y
02df	a2	0f	sbca	15,x	0375	e2	c0	sbc	0,pc
02e1	a2	4f	sbca	15,y	0377	e2	80	sbc	0,sp
02e3	a2	f0 10	sbca	16,sp	0379	e2	00	sbc	0,x
02e6	a2	e0 10	sbca	16,x	037b	e2	40	sbc	0,y
02e9	a2	e8 10	sbca	16,y	037d	e2	b0	sbc	1,sp+
02ec	a2	b7	sbca	8,sp+	037f	e2	30	sbc	1,x+
02ee	a2	37	sbca	8,x+	0381	e2	70	sbc	1,y+
02f0	a2	77	sbca	8,y+	0383	e2	81	sbc	1,sp
02f2	a2	b8	sbca	8,sp-	0385	e2	01	sbc	1,x
02f4	a2	38	sbca	8,x-	0387	e2	41	sbc	1,y
02f6	a2	78	sbca	8,y-	0389	e2	bf	sbc	1,sp-
02f8	a2	f4	sbca	a,sp	038b	e2	3f	sbc	1,x-
02fa	a2	e4	sbca	a,x	038d	e2	7f	sbc	1,y-
02fc	a2	ec	sbca	a,y	038f	e2	f8 7d	sbc	125,pc
02fe	a2	f5	sbca	b,sp	0392	e2	f0 7d	sbc	125,sp
0300	a2	e5	sbca	b,x	0395	e2	e0 7d	sbc	125,x
0302	a2	ed	sbca	b,y	0398	e2	e8 7d	sbc	125,y
0304	a2	f6	sbca	d,sp	039b	e2	8f	sbc	15,sp
0306	a2	e6	sbca	d,x	039d	e2	0f	sbc	15,x
0308	a2	ee	sbca	d,y	039f	e2	4f	sbc	15,y
030a	92	55	sbca	dir	03a1	e2	f0 10	sbc	16,sp
030c	92	55	sbca	dir	03a4	e2	e0 10	sbc	16,x
030e	b2	01 88	sbca	ext	03a7	e2	e8 10	sbc	16,y
0311	b2	01 88	sbca	ext	03aa	e2	b7	sbc	8,sp+
0314	a2	f2 01 88	sbca	ext,sp	03ac	e2	37	sbc	8,x+
0318	a2	e2 01 88	sbca	ext,x	03ae	e2	77	sbc	8,y+
031c	a2	ea 01 88	sbca	ext,y	03b0	e2	b8	sbc	8,sp-
0320	a2	f8 37	sbca	ind,pc	03b2	e2	38	sbc	8,x-
0323	a2	f0 37	sbca	ind,sp	03b4	e2	78	sbc	8,y-
0326	a2	e0 37	sbca	ind,x	03b6	e2	f4	sbc	a,sp
0329	a2	e8 37	sbca	ind,y	03b8	e2	e4	sbc	a,x
032c	a2	ce	sbca	small,pc	03ba	e2	ec	sbc	a,y
032e	a2	8e	sbca	small,sp	03bc	e2	f5	sbc	b,sp
0330	a2	0e	sbca	small,x	03be	e2	e5	sbc	b,x
0332	a2	4e	sbca	small,y	03c0	e2	ed	sbc	b,y
0334	c2	72	sbc	#immed	03c2	e2	f6	sbc	d,sp
0336	c2	72	sbc	#immed	03c4	e2	e6	sbc	d,x
0338	e2	a0	sbc	1,+sp	03c6	e2	ee	sbc	d,y
033a	e2	20	sbc	1,+x	03c8	d2	55	sbc	dir
033c	e2	60	sbc	1,+y	03ca	d2	55	sbc	dir
033e	e2	a7	sbc	8,+sp	03cc	f2	01 88	sbc	ext
0340	e2	27	sbc	8,+x	03cf	f2	01 88	sbc	ext
0342	e2	67	sbc	8,+y	03d2	e2	f2 01 88	sbc	ext,sp
0344	e2	c0	sbc	,pc	03d6	e2	e2 01 88	sbc	ext,x
0346	e2	80	sbc	,sp	03da	e2	ea 01 88	sbc	ext,y
0348	e2	00	sbc	,x	03de	e2	f8 37	sbc	ind,pc
034a	e2	40	sbc	,y	03e1	e2	f0 37	sbc	ind,sp
034c	e2	af	sbc	1,-sp	03e4	e2	e0 37	sbc	ind,x
034e	e2	2f	sbc	1,-x	03e7	e2	e8 37	sbc	ind,y
0350	e2	6f	sbc	1,-y	03ea	e2	ce	sbc	small,pc
0352	e2	a8	sbc	8,-sp	03ec	e2	8e	sbc	small,sp



03ee e2 0e	sbcb	small,x	0476 6a f0 7d	staa	125,sp
03f0 e2 4e	sbcb	small,y	0479 6a e0 7d	staa	125,x
03f2 14 01	sec		047c 6a e8 7d	staa	125,y
03f4 14 10	sei		047f 6a 8f	staa	15,sp
03f6 14 02	sev		0481 6a 0f	staa	15,x
03f8 b7 04	sex	a d	0483 6a 4f	staa	15,y
03fa b7 07	sex	a sp	0485 6a f0 10	staa	16,sp
03fc b7 07	sex	a,sp	0488 6a e0 10	staa	16,x
03fe b7 05	sex	a x	048b 6a e8 10	staa	16,y
0400 b7 05	sex	a,x	048e 6a b7	staa	8,sp+
0402 b7 06	sex	a y	0490 6a 37	staa	8,x+
0404 b7 06	sex	a,y	0492 6a 77	staa	8,y+
0406 b7 14	sex	b d	0494 6a b8	staa	8,sp-
0408 b7 17	sex	b sp	0496 6a 38	staa	8,x-
040a b7 17	sex	b,sp	0498 6a 78	staa	8,y-
040c b7 15	sex	b x	049a 6a f4	staa	a,sp
040e b7 15	sex	b,x	049c 6a e4	staa	a,x
0410 b7 16	sex	b y	049e 6a ec	staa	a,y
0412 b7 16	sex	b,y	04a0 6a f5	staa	b,sp
0414 b7 24	sex	ccr d	04a2 6a e5	staa	b,x
0416 b7 27	sex	ccr sp	04a4 6a ed	staa	b,y
0418 b7 25	sex	ccr x	04a6 6a f6	staa	d,sp
041a b7 26	sex	ccr y	04a8 6a e6	staa	d,x
041c 6a a0	staa	1,+sp	04aa 6a ee	staa	d,y
041e 6a 20	staa	1,+x	04ac 5a 55	staa	dir
0420 6a 60	staa	1,+y	04ae 5a 55	staa	dir
0422 6a a7	staa	8,+sp	04b0 7a 01 88	staa	ext
0424 6a 27	staa	8,+x	04b3 7a 01 88	staa	ext
0426 6a 67	staa	8,+y	04b6 6a f2 01 88	staa	ext,sp
0428 6a c0	staa	,pc	04ba 6a e2 01 88	staa	ext,x
042a 6a 80	staa	,sp	04be 6a ea 01 88	staa	ext,y
042c 6a 00	staa	,x	04c2 6a f8 37	staa	ind,pc
042e 6a 40	staa	,y	04c5 6a f0 37	staa	ind,sp
0430 6a af	staa	1,-sp	04c8 6a e0 37	staa	ind,x
0432 6a 2f	staa	1,-x	04cb 6a e8 37	staa	ind,y
0434 6a 6f	staa	1,-y	04ce 6a ce	staa	small,pc
0436 6a a8	staa	8,-sp	04d0 6a 8e	staa	small,sp
0438 6a 28	staa	8,-x	04d2 6a 0e	staa	small,x
043a 6a 68	staa	8,-y	04d4 6a 4e	staa	small,y
043c 6a 9f	staa	-1,sp	04d6 6b a0	stab	1,+sp
043e 6a 1f	staa	-1,x	04d8 6b 20	stab	1,+x
0440 6a 5f	staa	-1,y	04da 6b 60	stab	1,+y
0442 6a 90	staa	-16,sp	04dc 6b a7	stab	8,+sp
0444 6a 10	staa	-16,x	04de 6b 27	stab	8,+x
0446 6a 50	staa	-16,y	04e0 6b 67	stab	8,+y
0448 6a f1 ef	staa	-17,sp	04e2 6b c0	stab	,pc
044b 6a e1 ef	staa	-17,x	04e4 6b 80	stab	,sp
044e 6a e9 ef	staa	-17,y	04e6 6b 00	stab	,x
0451 6a d2	staa	-small,pc	04e8 6b 40	stab	,y
0453 6a 92	staa	-small,sp	04ea 6b af	stab	1,-sp
0455 6a 12	staa	-small,x	04ec 6b 2f	stab	1,-x
0457 6a 52	staa	-small,y	04ee 6b 6f	stab	1,-y
0459 6a c0	staa	0,pc	04f0 6b a8	stab	8,-sp
045b 6a 80	staa	0,sp	04f2 6b 28	stab	8,-x
045d 6a 00	staa	0,x	04f4 6b 68	stab	8,-y
045f 6a 40	staa	0,y	04f6 6b 9f	stab	-1,sp
0461 6a b0	staa	1,sp+	04f8 6b 1f	stab	-1,x
0463 6a 30	staa	1,x+	04fa 6b 5f	stab	-1,y
0465 6a 70	staa	1,y+	04fc 6b 90	stab	-16,sp
0467 6a 81	staa	1,sp	04fe 6b 10	stab	-16,x
0469 6a 01	staa	1,x	0500 6b 50	stab	-16,y
046b 6a 41	staa	1,y	0502 6b f1 ef	stab	-17,sp
046d 6a bf	staa	1,sp-	0505 6b e1 ef	stab	-17,x
046f 6a 3f	staa	1,x-	0508 6b e9 ef	stab	-17,y
0471 6a 7f	staa	1,y-	050b 6b d2	stab	-small,pc
0473 6a f8 7d	staa	125,pc	050d 6b 92	stab	-small,sp

050f 6b 12	stab	-small,x	05a6 6c 2f	std	1,-x
0511 6b 52	stab	-small,y	05a8 6c 6f	std	1,-y
0513 6b c0	stab	0,pc	05aa 6c a8	std	8,-sp
0515 6b 80	stab	0,sp	05ac 6c 28	std	8,-x
0517 6b 00	stab	0,x	05ae 6c 68	std	8,-y
0519 6b 40	stab	0,y	05b0 6c 9f	std	-1,sp
051b 6b b0	stab	1,sp+	05b2 6c 1f	std	-1,x
051d 6b 30	stab	1,x+	05b4 6c 5f	std	-1,y
051f 6b 70	stab	1,y+	05b6 6c 90	std	-16,sp
0521 6b 81	stab	1,sp	05b8 6c 10	std	-16,x
0523 6b 01	stab	1,x	05ba 6c 50	std	-16,y
0525 6b 41	stab	1,y	05bc 6c f1 ef	std	-17,sp
0527 6b bf	stab	1,sp-	05bf 6c e1 ef	std	-17,x
0529 6b 3f	stab	1,x-	05c2 6c e9 ef	std	-17,y
052b 6b 7f	stab	1,y-	05c5 6c d2	std	-small,pc
052d 6b f8 7d	stab	125,pc	05c7 6c 92	std	-small,sp
0530 6b f0 7d	stab	125,sp	05c9 6c 12	std	-small,x
0533 6b e0 7d	stab	125,x	05cb 6c 52	std	-small,y
0536 6b e8 7d	stab	125,y	05cd 6c c0	std	0,pc
0539 6b 8f	stab	15,sp	05cf 6c 80	std	0,sp
053b 6b 0f	stab	15,x	05d1 6c 00	std	0,x
053d 6b 4f	stab	15,y	05d3 6c 40	std	0,y
053f 6b f0 10	stab	16,sp	05d5 6c b0	std	1,sp+
0542 6b e0 10	stab	16,x	05d7 6c 30	std	1,x+
0545 6b e8 10	stab	16,y	05d9 6c 70	std	1,y+
0548 6b b7	stab	8,sp+	05db 6c 81	std	1,sp
054a 6b 37	stab	8,x+	05dd 6c 01	std	1,x
054c 6b 77	stab	8,y+	05df 6c 41	std	1,y
054e 6b b8	stab	8,sp-	05e1 6c bf	std	1,sp-
0550 6b 38	stab	8,x-	05e3 6c 3f	std	1,x-
0552 6b 78	stab	8,y-	05e5 6c 7f	std	1,y-
0554 6b f4	stab	a,sp	05e7 6c f8 7d	std	125,pc
0556 6b e4	stab	a,x	05ea 6c f0 7d	std	125,sp
0558 6b ec	stab	a,y	05ed 6c e0 7d	std	125,x
055a 6b f5	stab	b,sp	05f0 6c e8 7d	std	125,y
055c 6b e5	stab	b,x	05f3 6c 8f	std	15,sp
055e 6b ed	stab	b,y	05f5 6c 0f	std	15,x
0560 6b f6	stab	d,sp	05f7 6c 4f	std	15,y
0562 6b e6	stab	d,x	05f9 6c f0 10	std	16,sp
0564 6b ee	stab	d,y	05fc 6c e0 10	std	16,x
0566 5b 55	stab	dir	05ff 6c e8 10	std	16,y
0568 5b 55	stab	dir	0602 6c b7	std	8,sp+
056a 7b 01 88	stab	ext	0604 6c 37	std	8,x+
056d 7b 01 88	stab	ext	0606 6c 77	std	8,y+
0570 6b f2 01 88	stab	ext,sp	0608 6c b8	std	8,sp-
0574 6b e2 01 88	stab	ext,x	060a 6c 38	std	8,x-
0578 6b ea 01 88	stab	ext,y	060c 6c 78	std	8,y-
057c 6b f8 37	stab	ind,pc	060e 6c f4	std	a,sp
057f 6b f0 37	stab	ind,sp	0610 6c e4	std	a,x
0582 6b e0 37	stab	ind,x	0612 6c ec	std	a,y
0585 6b e8 37	stab	ind,y	0614 6c f5	std	b,sp
0588 6b ce	stab	small,pc	0616 6c e5	std	b,x
058a 6b 8e	stab	small,sp	0618 6c ed	std	b,y
058c 6b 0e	stab	small,x	061a 6c f6	std	d,sp
058e 6b 4e	stab	small,y	061c 6c e6	std	d,x
0590 6c a0	std	1,+sp	061e 6c ee	std	d,y
0592 6c 20	std	1,+x	0620 5c 55	std	dir
0594 6c 60	std	1,+y	0622 5c 55	std	dir
0596 6c a7	std	8,+sp	0624 7c 01 88	std	ext
0598 6c 27	std	8,+x	0627 7c 01 88	std	ext
059a 6c 67	std	8,+y	062a 6c f2 01 88	std	ext,sp
059c 6c c0	std	,pc	062e 6c e2 01 88	std	ext,x
059e 6c 80	std	,sp	0632 6c ea 01 88	std	ext,y
05a0 6c 00	std	,x	0636 6c f8 37	std	ind,pc
05a2 6c 40	std	,y	0639 6c f0 37	std	ind,sp
05a4 6c af	std	1,-sp	063c 6c e0 37	std	ind,x

063f	6c	e8	37	std	ind,y	06ce	6f	ec	sts	a,y
0642	6c	ce		std	small,pc	06d0	6f	f5	sts	b,sp
0644	6c	8e		std	small,sp	06d2	6f	e5	sts	b,x
0646	6c	0e		std	small,x	06d4	6f	ed	sts	b,y
0648	6c	4e		std	small,y	06d6	6f	f6	sts	d,sp
064a	18	3e		stop		06d8	6f	e6	sts	d,x
064c	6f	a0		sts	1,+sp	06da	6f	ee	sts	d,y
064e	6f	20		sts	1,+x	06dc	5f	55	sts	dir
0650	6f	60		sts	1,+y	06de	7f	01 88	sts	ext
0652	6f	a7		sts	8,+sp	06e1	6f	f2 01 88	sts	ext,sp
0654	6f	27		sts	8,+x	06e5	6f	e2 01 88	sts	ext,x
0656	6f	67		sts	8,+y	06e9	6f	ea 01 88	sts	ext,y
0658	6f	c0		sts	,pc	06ed	6f	f8 37	sts	ind,pc
065a	6f	80		sts	,sp	06f0	6f	f0 37	sts	ind,sp
065c	6f	00		sts	,x	06f3	6f	e0 37	sts	ind,x
065e	6f	40		sts	,y	06f6	6f	e8 37	sts	ind,y
0660	6f	af		sts	1,-sp	06f9	6f	ce	sts	small,pc
0662	6f	2f		sts	1,-x	06fb	6f	8e	sts	small,sp
0664	6f	6f		sts	1,-y	06fd	6f	0e	sts	small,x
0666	6f	a8		sts	8,-sp	06ff	6f	4e	sts	small,y
0668	6f	28		sts	8,-x	0701	6e	a0	stx	1,+sp
066a	6f	68		sts	8,-y	0703	6e	20	stx	1,+x
066c	6f	9f		sts	-1,sp	0705	6e	60	stx	1,+y
066e	6f	1f		sts	-1,x	0707	6e	a7	stx	8,+sp
0670	6f	5f		sts	-1,y	0709	6e	27	stx	8,+x
0672	6f	90		sts	-16,sp	070b	6e	67	stx	8,+y
0674	6f	10		sts	-16,x	070d	6e	c0	stx	,pc
0676	6f	50		sts	-16,y	070f	6e	80	stx	,sp
0678	6f	f1 ef		sts	-17,sp	0711	6e	00	stx	,x
067b	6f	e1 ef		sts	-17,x	0713	6e	40	stx	,y
067e	6f	e9 ef		sts	-17,y	0715	6e	af	stx	1,-sp
0681	6f	d2		sts	-small,pc	0717	6e	2f	stx	1,-x
0683	6f	92		sts	-small,sp	0719	6e	6f	stx	1,-y
0685	6f	12		sts	-small,x	071b	6e	a8	stx	8,-sp
0687	6f	52		sts	-small,y	071d	6e	28	stx	8,-x
0689	6f	c0		sts	0,pc	071f	6e	68	stx	8,-y
068b	6f	80		sts	0,sp	0721	6e	9f	stx	-1,sp
068d	6f	00		sts	0,x	0723	6e	1f	stx	-1,x
068f	6f	40		sts	0,y	0725	6e	5f	stx	-1,y
0691	6f	b0		sts	1,sp+	0727	6e	90	stx	-16,sp
0693	6f	30		sts	1,x+	0729	6e	10	stx	-16,x
0695	6f	70		sts	1,y+	072b	6e	50	stx	-16,y
0697	6f	81		sts	1,sp	072d	6e	f1 ef	stx	-17,sp
0699	6f	01		sts	1,x	0730	6e	e1 ef	stx	-17,x
069b	6f	41		sts	1,y	0733	6e	e9 ef	stx	-17,y
069d	6f	bf		sts	1,sp-	0736	6e	d2	stx	-small,pc
069f	6f	3f		sts	1,x-	0738	6e	92	stx	-small,sp
06a1	6f	7f		sts	1,y-	073a	6e	12	stx	-small,x
06a3	6f	f8 7d		sts	125,pc	073c	6e	52	stx	-small,y
06a6	6f	f0 7d		sts	125,sp	073e	6e	c0	stx	0,pc
06a9	6f	e0 7d		sts	125,x	0740	6e	80	stx	0,sp
06ac	6f	e8 7d		sts	125,y	0742	6e	00	stx	0,x
06af	6f	8f		sts	15,sp	0744	6e	40	stx	0,y
06b1	6f	0f		sts	15,x	0746	6e	b0	stx	1,sp+
06b3	6f	4f		sts	15,y	0748	6e	30	stx	1,x+
06b5	6f	f0 10		sts	16,sp	074a	6e	70	stx	1,y+
06b8	6f	e0 10		sts	16,x	074c	6e	81	stx	1,sp
06bb	6f	e8 10		sts	16,y	074e	6e	01	stx	1,x
06be	6f	b7		sts	8,sp+	0750	6e	41	stx	1,y
06c0	6f	37		sts	8,x+	0752	6e	bf	stx	1,sp-
06c2	6f	77		sts	8,y+	0754	6e	3f	stx	1,x-
06c4	6f	b8		sts	8,sp-	0756	6e	7f	stx	1,y-
06c6	6f	38		sts	8,x-	0758	6e	f8 7d	stx	125,pc
06c8	6f	78		sts	8,y-	075b	6e	f0 7d	stx	125,sp
06ca	6f	f4		sts	a,sp	075e	6e	e0 7d	stx	125,x
06cc	6f	e4		sts	a,x	0761	6e	e8 7d	stx	125,y

0764 6e 8f	stx	15,sp	07fa 6d 80	sty	0,sp
0766 6e 0f	stx	15,x	07fc 6d 00	sty	0,x
0768 6e 4f	stx	15,y	07fe 6d 40	sty	0,y
076a 6e f0 10	stx	16,sp	0800 6d b0	sty	1,sp+
076d 6e e0 10	stx	16,x	0802 6d 30	sty	1,x+
0770 6e e8 10	stx	16,y	0804 6d 70	sty	1,y+
0773 6e b7	stx	8,sp+	0806 6d 81	sty	1,sp
0775 6e 37	stx	8,x+	0808 6d 01	sty	1,x
0777 6e 77	stx	8,y+	080a 6d 41	sty	1,y
0779 6e b8	stx	8,sp-	080c 6d bf	sty	1,sp-
077b 6e 38	stx	8,x-	080e 6d 3f	sty	1,x-
077d 6e 78	stx	8,y-	0810 6d 7f	sty	1,y-
077f 6e f4	stx	a,sp	0812 6d f8 7d	sty	125,pc
0781 6e e4	stx	a,x	0815 6d f0 7d	sty	125,sp
0783 6e ec	stx	a,y	0818 6d e0 7d	sty	125,x
0785 6e f5	stx	b,sp	081b 6d e8 7d	sty	125,y
0787 6e e5	stx	b,x	081e 6d 8f	sty	15,sp
0789 6e ed	stx	b,y	0820 6d 0f	sty	15,x
078b 6e f6	stx	d,sp	0822 6d 4f	sty	15,y
078d 6e e6	stx	d,x	0824 6d f0 10	sty	16,sp
078f 6e ee	stx	d,y	0827 6d e0 10	sty	16,x
0791 5e 55	stx	dir	082a 6d e8 10	sty	16,y
0793 5e 55	stx	dir	082d 6d b7	sty	8,sp+
0795 7e 01 88	stx	ext	082f 6d 37	sty	8,x+
0798 7e 01 88	stx	ext	0831 6d 77	sty	8,y+
079b 6e f2 01 88	stx	ext,sp	0833 6d b8	sty	8,sp-
079f 6e e2 01 88	stx	ext,x	0835 6d 38	sty	8,x-
07a3 6e ea 01 88	stx	ext,y	0837 6d 78	sty	8,y-
07a7 6e f8 37	stx	ind,pc	0839 6d f4	sty	a,sp
07aa 6e f0 37	stx	ind,sp	083b 6d e4	sty	a,x
07ad 6e e0 37	stx	ind,x	083d 6d ec	sty	a,y
07b0 6e e8 37	stx	ind,y	083f 6d f5	sty	b,sp
07b3 6e ce	stx	small,pc	0841 6d e5	sty	b,x
07b5 6e 8e	stx	small,sp	0843 6d ed	sty	b,y
07b7 6e 0e	stx	small,x	0845 6d f6	sty	d,sp
07b9 6e 4e	stx	small,y	0847 6d e6	sty	d,x
07bb 6d a0	sty	1,+sp	0849 6d ee	sty	d,y
07bd 6d 20	sty	1,+x	084b 5d 55	sty	dir
07bf 6d 60	sty	1,+y	084d 5d 55	sty	dir
07c1 6d a7	sty	8,+sp	084f 7d 01 88	sty	ext
07c3 6d 27	sty	8,+x	0852 7d 01 88	sty	ext
07c5 6d 67	sty	8,+y	0855 6d f2 01 88	sty	ext,sp
07c7 6d c0	sty	,pc	0859 6d e2 01 88	sty	ext,x
07c9 6d 80	sty	,sp	085d 6d ea 01 88	sty	ext,y
07cb 6d 00	sty	,x	0861 6d f8 37	sty	ind,pc
07cd 6d 40	sty	,y	0864 6d f0 37	sty	ind,sp
07cf 6d af	sty	1,-sp	0867 6d e0 37	sty	ind,x
07d1 6d 2f	sty	1,-x	086a 6d e8 37	sty	ind,y
07d3 6d 6f	sty	1,-y	086d 6d ce	sty	small,pc
07d5 6d a8	sty	8,-sp	086f 6d 8e	sty	small,sp
07d7 6d 28	sty	8,-x	0871 6d 0e	sty	small,x
07d9 6d 68	sty	8,-y	0873 6d 4e	sty	small,y
07db 6d 9f	sty	-1,sp	0875 80 72	suba	#immed
07dd 6d 1f	sty	-1,x	0877 a0 a0	suba	1,+sp
07df 6d 5f	sty	-1,y	0879 a0 20	suba	1,+x
07e1 6d 90	sty	-16,sp	087b a0 60	suba	1,+y
07e3 6d 10	sty	-16,x	087d a0 a7	suba	8,+sp
07e5 6d 50	sty	-16,y	087f a0 27	suba	8,+x
07e7 6d f1 ef	sty	-17,sp	0881 a0 67	suba	8,+y
07ea 6d e1 ef	sty	-17,x	0883 a0 c0	suba	,pc
07ed 6d e9 ef	sty	-17,y	0885 a0 80	suba	,sp
07f0 6d d2	sty	-small,pc	0887 a0 00	suba	,x
07f2 6d 92	sty	-small,sp	0889 a0 40	suba	,y
07f4 6d 12	sty	-small,x	088b a0 af	suba	1,-sp
07f6 6d 52	sty	-small,y	088d a0 2f	suba	1,-x
07f8 6d c0	sty	0,pc	088f a0 6f	suba	1,-y

0891 a0 a8	suba	8,-sp	092a a0 4e	suba	small,y
0893 a0 28	suba	8,-x	092c c0 72	subb	#immed
0895 a0 68	suba	8,-y	092e c0 72	subb	#immed
0897 a0 9f	suba	-1,sp	0930 e0 a0	subb	1,+sp
0899 a0 1f	suba	-1,x	0932 e0 20	subb	1,+x
089b a0 5f	suba	-1,y	0934 e0 60	subb	1,+y
089d a0 90	suba	-16,sp	0936 e0 a7	subb	8,+sp
089f a0 10	suba	-16,x	0938 e0 27	subb	8,+x
08a1 a0 50	suba	-16,y	093a e0 67	subb	8,+y
08a3 a0 f1 ef	suba	-17,sp	093c e0 c0	subb	,pc
08a6 a0 e1 ef	suba	-17,x	093e e0 80	subb	,sp
08a9 a0 e9 ef	suba	-17,y	0940 e0 00	subb	,x
08ac a0 d2	suba	-small,pc	0942 e0 40	subb	,y
08ae a0 92	suba	-small,sp	0944 e0 af	subb	1,-sp
08b0 a0 12	suba	-small,x	0946 e0 2f	subb	1,-x
08b2 a0 52	suba	-small,y	0948 e0 6f	subb	1,-y
08b4 a0 c0	suba	0,pc	094a e0 a8	subb	8,-sp
08b6 a0 80	suba	0,sp	094c e0 28	subb	8,-x
08b8 a0 00	suba	0,x	094e e0 68	subb	8,-y
08ba a0 40	suba	0,y	0950 e0 9f	subb	-1,sp
08bc a0 b0	suba	1,sp+	0952 e0 1f	subb	-1,x
08be a0 30	suba	1,x+	0954 e0 5f	subb	-1,y
08c0 a0 70	suba	1,y+	0956 e0 90	subb	-16,sp
08c2 a0 81	suba	1,sp	0958 e0 10	subb	-16,x
08c4 a0 01	suba	1,x	095a e0 50	subb	-16,y
08c6 a0 41	suba	1,y	095c e0 f1 ef	subb	-17,sp
08c8 a0 bf	suba	1,sp-	095f e0 e1 ef	subb	-17,x
08ca a0 3f	suba	1,x-	0962 e0 e9 ef	subb	-17,y
08cc a0 7f	suba	1,y-	0965 e0 d2	subb	-small,pc
08ce a0 f8 7d	suba	125,pc	0967 e0 92	subb	-small,sp
08d1 a0 f0 7d	suba	125,sp	0969 e0 12	subb	-small,x
08d4 a0 e0 7d	suba	125,x	096b e0 52	subb	-small,y
08d7 a0 e8 7d	suba	125,y	096d e0 c0	subb	0,pc
08da a0 8f	suba	15,sp	096f e0 80	subb	0,sp
08dc a0 0f	suba	15,x	0971 e0 00	subb	0,x
08de a0 4f	suba	15,y	0973 e0 40	subb	0,y
08e0 a0 f0 10	suba	16,sp	0975 e0 b0	subb	1,sp+
08e3 a0 e0 10	suba	16,x	0977 e0 30	subb	1,x+
08e6 a0 e8 10	suba	16,y	0979 e0 70	subb	1,y+
08e9 a0 b7	suba	8,sp+	097b e0 81	subb	1,sp
08eb a0 37	suba	8,x+	097d e0 01	subb	1,x
08ed a0 77	suba	8,y+	097f e0 41	subb	1,y
08ef a0 b8	suba	8,sp-	0981 e0 bf	subb	1,sp-
08f1 a0 38	suba	8,x-	0983 e0 3f	subb	1,x-
08f3 a0 78	suba	8,y-	0985 e0 7f	subb	1,y-
08f5 a0 f4	suba	a,sp	0987 e0 f8 7d	subb	125,pc
08f7 a0 e4	suba	a,x	098a e0 f0 7d	subb	125,sp
08f9 a0 ec	suba	a,y	098d e0 e0 7d	subb	125,x
08fb a0 f5	suba	b,sp	0990 e0 e8 7d	subb	125,y
08fd a0 e5	suba	b,x	0993 e0 8f	subb	15,sp
08ff a0 ed	suba	b,y	0995 e0 0f	subb	15,x
0901 a0 f6	suba	d,sp	0997 e0 4f	subb	15,y
0903 a0 e6	suba	d,x	0999 e0 f0 10	subb	16,sp
0905 a0 ee	suba	d,y	099c e0 e0 10	subb	16,x
0907 90 55	suba	dir	099f e0 e8 10	subb	16,y
0909 b0 01 88	suba	ext	09a2 e0 b7	subb	8,sp+
090c a0 f2 01 88	suba	ext,sp	09a4 e0 37	subb	8,x+
0910 a0 e2 01 88	suba	ext,x	09a6 e0 77	subb	8,y+
0914 a0 ea 01 88	suba	ext,y	09a8 e0 b8	subb	8,sp-
0918 a0 f8 37	suba	ind,pc	09aa e0 38	subb	8,x-
091b a0 f0 37	suba	ind,sp	09ac e0 78	subb	8,y-
091e a0 e0 37	suba	ind,x	09ae e0 f4	subb	a,sp
0921 a0 e8 37	suba	ind,y	09b0 e0 e4	subb	a,x
0924 a0 ce	suba	small,pc	09b2 e0 ec	subb	a,y
0926 a0 8e	suba	small,sp	09b4 e0 f5	subb	b,sp
0928 a0 0e	suba	small,x	09b6 e0 e5	subb	b,x

09b8 e0 ed	subb	b,y	0a50 a3 e8 7d	subd	125,y
09ba e0 f6	subb	d,sp	0a53 a3 8f	subd	15,sp
09bc e0 e6	subb	d,x	0a55 a3 0f	subd	15,x
09be e0 ee	subb	d,y	0a57 a3 4f	subd	15,y
09c0 d0 55	subb	dir	0a59 a3 f0 10	subd	16,sp
09c2 d0 55	subb	dir	0a5c a3 e0 10	subd	16,x
09c4 f0 01 88	subb	ext	0a5f a3 e8 10	subd	16,y
09c7 f0 01 88	subb	ext	0a62 a3 b7	subd	8,sp+
09ca e0 f2 01 88	subb	ext,sp	0a64 a3 37	subd	8,x+
09ce e0 e2 01 88	subb	ext,x	0a66 a3 77	subd	8,y+
09d2 e0 ea 01 88	subb	ext,y	0a68 a3 b8	subd	8,sp-
09d6 e0 f8 37	subb	ind,pc	0a6a a3 38	subd	8,x-
09d9 e0 f0 37	subb	ind,sp	0a6c a3 78	subd	8,y-
09dc e0 f0 37	subb	ind,x	0a6e a3 f4	subd	a,sp
09df e0 e8 37	subb	ind,y	0a70 a3 e4	subd	a,x
09e2 e0 ce	subb	small,pc	0a72 a3 ec	subd	a,y
09e4 e0 8e	subb	small,sp	0a74 a3 f5	subd	b,sp
09e6 e0 0e	subb	small,x	0a76 a3 e5	subd	b,x
09e8 e0 4e	subb	small,y	0a78 a3 ed	subd	b,y
09ea 83 00 72	subd	#immed	0a7a a3 f6	subd	d,sp
09ed 83 00 72	subd	#immed	0a7c a3 e6	subd	d,x
09f0 a3 a0	subd	1,+sp	0a7e a3 ee	subd	d,y
09f2 a3 20	subd	1,+x	0a80 93 55	subd	dir
09f4 a3 60	subd	1,+y	0a82 93 55	subd	dir
09f6 a3 a7	subd	8,+sp	0a84 b3 01 88	subd	ext
09f8 a3 27	subd	8,+x	0a87 b3 01 88	subd	ext
09fa a3 67	subd	8,+y	0a8a a3 f2 01 88	subd	ext,sp
09fc a3 c0	subd	,pc	0a8e a3 e2 01 88	subd	ext,x
09fe a3 80	subd	,sp	0a92 a3 ea 01 88	subd	ext,y
0a00 a3 00	subd	,x	0a96 a3 f8 37	subd	ind,pc
0a02 a3 40	subd	,y	0a99 a3 f0 37	subd	ind,sp
0a04 a3 af	subd	1,-sp	0a9c a3 e0 37	subd	ind,x
0a06 a3 2f	subd	1,-x	0a9f a3 e8 37	subd	ind,y
0a08 a3 6f	subd	1,-y	0aa2 a3 ce	subd	small,pc
0a0a a3 a8	subd	8,-sp	0aa4 a3 8e	subd	small,sp
0a0c a3 28	subd	8,-x	0aa6 a3 0e	subd	small,x
0a0e a3 68	subd	8,-y	0aa8 a3 4e	subd	small,y
0a10 a3 9f	subd	-1,sp	0aaa 3f	swi	
0a12 a3 1f	subd	-1,x	0aab b7 c4	swpb	d
0a14 a3 5f	subd	-1,y	0aad 18 0e	tab	
0a16 a3 90	subd	-16,sp	0aaf b7 02	tap	
0a18 a3 10	subd	-16,x	0ab1 18 0f	tba	
0a1a a3 50	subd	-16,y	0ab3 18 3d e5	tbl	b,x
0a1c a3 f1 ef	subd	-17,sp	0ab6 b7 00	tfr	a a
0a1f a3 e1 ef	subd	-17,x	0ab8 b7 00	tfr	a,a
0a22 a3 e9 ef	subd	-17,y	0aba b7 01	tfr	a b
0a25 a3 d2	subd	-small,pc	0abc b7 01	tfr	a,b
0a27 a3 92	subd	-small,sp	0abe b7 02	tfr	a ccr
0a29 a3 12	subd	-small,x	0ac0 b7 04	tfr	a d
0a2b a3 52	subd	-small,y	0ac2 b7 07	tfr	a sp
0a2d a3 c0	subd	0,pc	0ac4 b7 05	tfr	a x
0a2f a3 80	subd	0,sp	0ac6 b7 05	tfr	a,x
0a31 a3 00	subd	0,x	0ac8 b7 06	tfr	a y
0a33 a3 40	subd	0,y	0aca b7 06	tfr	a,y
0a35 a3 b0	subd	1,sp+	0acc b7 10	tfr	b a
0a37 a3 30	subd	1,x+	0ace b7 11	tfr	b b
0a39 a3 70	subd	1,y+	0ad0 b7 12	tfr	b ccr
0a3b a3 81	subd	1,sp	0ad2 b7 14	tfr	b d
0a3d a3 01	subd	1,x	0ad4 b7 17	tfr	b sp
0a3f a3 41	subd	1,y	0ad6 b7 15	tfr	b x
0a41 a3 bf	subd	1,sp-	0ad8 b7 16	tfr	b y
0a43 a3 3f	subd	1,x-	0ada b7 20	tfr	ccr a
0a45 a3 7f	subd	1,y-	0adc b7 21	tfr	ccr b
0a47 a3 f8 7d	subd	125,pc	0ade b7 22	tfr	ccr ccr
0a4a a3 f0 7d	subd	125,sp	0ae0 b7 24	tfr	ccr d
0a4d a3 e0 7d	subd	125,x	0ae2 b7 27	tfr	ccr sp



0ae4 b7 25	tfr	ccr x	0b6b e7 70	tst	1,y+
0ae6 b7 26	tfr	ccr y	0b6d e7 81	tst	1,sp
0ae8 b7 40	tfr	d a	0b6f e7 01	tst	1,x
0aea b7 41	tfr	d b	0b71 e7 41	tst	1,y
0aec b7 42	tfr	d ccr	0b73 e7 bf	tst	1,sp-
0aee b7 44	tfr	d d	0b75 e7 3f	tst	1,x-
0af0 b7 47	tfr	d sp	0b77 e7 7f	tst	1,y-
0af2 b7 45	tfr	d x	0b79 e7 f8 7d	tst	125,pc
0af4 b7 46	tfr	d y	0b7c e7 f0 7d	tst	125,sp
0af6 b7 70	tfr	sp a	0b7f e7 e0 7d	tst	125,x
0af8 b7 71	tfr	sp b	0b82 e7 e8 7d	tst	125,y
0afa b7 72	tfr	sp ccr	0b85 e7 8f	tst	15,sp
0afc b7 74	tfr	sp d	0b87 e7 0f	tst	15,x
0afe b7 77	tfr	sp sp	0b89 e7 4f	tst	15,y
0b00 b7 75	tfr	sp x	0b8b e7 f0 10	tst	16,sp
0b02 b7 76	tfr	sp y	0b8e e7 e0 10	tst	16,x
0b04 b7 50	tfr	x a	0b91 e7 e8 10	tst	16,y
0b06 b7 51	tfr	x b	0b94 e7 b7	tst	8,sp+
0b08 b7 52	tfr	x ccr	0b96 e7 37	tst	8,x+
0b0a b7 54	tfr	x d	0b98 e7 77	tst	8,y+
0b0c b7 57	tfr	x sp	0b9a e7 b8	tst	8,sp-
0b0e b7 55	tfr	x x	0b9c e7 38	tst	8,x-
0b10 b7 56	tfr	x y	0b9e e7 78	tst	8,y-
0b12 b7 60	tfr	y a	0ba0 e7 f4	tst	a,sp
0b14 b7 61	tfr	y b	0ba2 e7 e4	tst	a,x
0b16 b7 62	tfr	y ccr	0ba4 e7 ec	tst	a,y
0b18 b7 64	tfr	y d	0ba6 e7 f5	tst	b,sp
0b1a b7 67	tfr	y sp	0ba8 e7 e5	tst	b,x
0b1c b7 65	tfr	y x	0baa e7 ed	tst	b,y
0b1e b7 66	tfr	y y	0bac e7 f6	tst	d,sp
0b20 b7 20	tpa		0bae e7 e6	tst	d,x
0b22 e7 a0	tst	1,+sp	0bb0 e7 ee	tst	d,y
0b24 e7 20	tst	1,+x	0bb2 f7 00 55	tst	dir
0b26 e7 60	tst	1,y	0bb5 f7 01 88	tst	ext
0b28 e7 a7	tst	8,+sp	0bb8 f7 01 88	tst	ext
0b2a e7 27	tst	8,+x	0bbb e7 f2 01 88	tst	ext,sp
0b2c e7 67	tst	8,+y	0bbf e7 e2 01 88	tst	ext,x
0b2e e7 c0	tst	,pc	0bc3 e7 ea 01 88	tst	ext,y
0b30 e7 80	tst	,sp	0bc7 e7 f8 37	tst	ind,pc
0b32 e7 00	tst	,x	0bca e7 f0 37	tst	ind,sp
0b34 e7 40	tst	,y	0bcd e7 e0 37	tst	ind,x
0b36 e7 af	tst	1,-sp	0bd0 e7 e8 37	tst	ind,y
0b38 e7 2f	tst	1,-x	0bd3 e7 ce	tst	small,pc
0b3a e7 6f	tst	1,-y	0bd5 e7 8e	tst	small,sp
0b3c e7 a8	tst	8,-sp	0bd7 e7 0e	tst	small,x
0b3e e7 28	tst	8,-x	0bd9 e7 4e	tst	small,y
0b40 e7 68	tst	8,-y	0bdb 97	tsta	
0b42 e7 9f	tst	-1,sp	0bdc d7	tstb	
0b44 e7 1f	tst	-1,x	0bdd b7 75	tsx	
0b46 e7 5f	tst	-1,y	0bdf b7 76	tsy	
0b48 e7 90	tst	-16,sp	0bel b7 57	txs	
0b4a e7 10	tst	-16,x	0be3 b7 67	tys	
0b4c e7 50	tst	-16,y	0be5 18 39	trap	\$39
0b4e e7 f1 ef	tst	-17,sp	0be7 3e	wai	
0b51 e7 e1 ef	tst	-17,x	0be8 18 3c	wav	
0b54 e7 e9 ef	tst	-17,y	0bea b7 c5	xgdx	
0b57 e7 d2	tst	-small,pc	0bec b7 c6	xgdy	
0b59 e7 92	tst	-small,sp	0bee 39	pshc	
0b5b e7 12	tst	-small,x	0bef 0a	rtc	
0b5d e7 52	tst	-small,y			
0b5f e7 c0	tst	0,pc			
0b61 e7 80	tst	0,sp			
0b63 e7 00	tst	0,x			
0b65 e7 40	tst	0,y			
0b67 e7 b0	tst	1,sp+			
0b69 e7 30	tst	1,x+			

## INDEX

### A

ABA instruction 6-8  
 Abbreviations for system resources 1-2  
 ABX instruction 6-9  
 ABY instruction 6-10  
 Accumulator direct indexed addressing mode 3-9  
 Accumulator offset indexed addressing mode 3-9  
 Accumulators 2-1, 5-8, 5-19  
   A 2-1, 3-5, 5-8, 6-8, 6-11, 6-13, 6-15 to 6-16,  
     6-20, 6-24, 6-35, 6-53, 6-57, 6-60, 6-63,  
     6-69 to 6-71, 6-73, 6-87, 6-90, 6-92 to 6-93,  
     6-97, 6-122, 6-124, 6-132, 6-134, 6-136,  
     6-139 to 6-140, 6-142 to 6-143, 6-146,  
     6-148, 6-151, 6-154, 6-157, 6-160, 6-167,  
     6-169, 6-171, 6-174, 6-177, 6-179 to 6-180,  
     6-185 to 6-186, 6-193, 6-196 to 6-204, 6-207  
   B 2-1, 3-5, 5-8, 6-8 to 6-10, 6-12, 6-14 to 6-15,  
     6-17, 6-21, 6-25, 6-36, 6-53, 6-58, 6-61, 6-64,  
     6-70 to 6-71, 6-74, 6-88 to 6-90,  
     6-92 to 6-93, 6-98, 6-123 to 6-124, 6-133,  
     6-137, 6-146, 6-149, 6-152, 6-155, 6-161,  
     6-172, 6-175, 6-177, 6-179, 6-181, 6-185,  
     6-187, 6-194, 6-196 to 6-197,  
     6-199 to 6-203, 6-208  
   D 2-1, 3-5, 5-8, 6-15, 6-22, 6-65, 6-70 to 6-71,  
     6-78 to 6-79, 6-81 to 6-86, 6-89 to 6-95,  
     6-124, 6-134, 6-138, 6-146, 6-157, 6-163,  
     6-185, 6-188, 6-195 to 6-196, 6-200,  
     6-202 to 6-203, 6-215 to 6-216  
 Indexed addressing 3-9  
 ADCA instruction 6-11  
 ADCB instruction 6-12  
 ADDA instruction 6-13  
 AADB instruction 6-14  
 ADDD instruction 6-15  
 Addition instructions 5-3, 6-8 to 6-15  
 ADDR mnemonic 1-3  
 Addressing modes 3-1  
   Direct 3-3  
   Extended 3-3  
   Immediate 3-2  
   Indexed 2-2, 3-5  
   Inherent 3-2  
   Memory expansion 10-7  
   Relative 3-4  
 ANDA instruction 6-16  
 ANDB instruction 6-17  
 ANDCC instruction 6-18  
 ASL instruction 6-19  
 ASLA instruction 6-20  
 ASLB instruction 6-21  
 ASLD instruction 6-22  
 ASR instruction 6-23

ASRA instruction 6-24  
 ASRB instruction 6-25  
 Asserted 1-3  
 Automatic indexing 3-8  
 Automatic program stack 2-2

### B

Background debugging mode 5-22, 8-6  
 BKGD pin 8-7 to 8-9  
 Commands 8-9 to 8-10  
 Enabling and disabling 8-6  
 Instruction 5-22, 6-31, 8-6  
 Registers 8-11  
 ROM 8-6  
 Serial interface 8-7 to 8-9  
 Base index register 3-6, 3-10  
 BCC instruction 6-26  
 BCLR instruction 6-27  
 BCS instruction 6-28  
 BEQ instruction 6-29  
 BGE instruction 6-30  
 BGND instruction 5-22, 6-31, 8-6  
 BGT instruction 6-32  
 BHI instruction 6-33  
 BHS instruction 6-34  
 Binary-coded decimal instructions 5-4, 6-8,  
   6-11 to 6-14, 6-69  
 Bit manipulation instructions 5-7, 6-27, 6-48, B-15,  
   C-1  
   Mask operand 3-11, 6-27, 6-48  
   Multiple addressing modes 3-11, 6-27, 6-48  
 Bit test instructions 5-7, 6-35 to 6-36, C-1  
 BITA instruction 6-35  
 BITB instruction 6-36  
 Bit-condition branches 5-16, 6-45, 6-47  
 BKGD pin 8-7 to 8-9  
 BLE instruction 6-37  
 BLO instruction 6-38  
 BLS instruction 6-39  
 BLT instruction 6-40  
 BMI instruction 6-41  
 BNE instruction 6-42  
 Boolean logic instructions 5-6  
   AND 6-16 to 6-18  
   Complement 6-62 to 6-64  
   Exclusive OR 6-87 to 6-88  
   Inclusive OR 6-151 to 6-153  
   Negate 6-147 to 6-149  
 BPL instruction 6-43  
 BRA instruction 6-44  
 Branch instructions 3-4, 4-4 to 4-5, 5-13, C-4  
   Bit-condition 4-4 to 4-5, 5-16, 6-45, 6-47  
   Long 4-4 to 4-5, 5-13, 6-104 to 6-121, B-13



# INDEX

Loop primitive 4-5, 5-16, 6-70 to 6-71,  
     6-92 to 6-93, 6-200, 6-202  
 Offset values 5-13, 5-16  
 Offsets 3-4  
 Short 4-4 to 4-5, 5-13, 6-26, 6-28 to 6-30,  
     6-32 to 6-34, 6-37 to 6-44, 6-46, 6-50 to 6-51  
 Signed 5-13, 6-30, 6-32, 6-37, 6-40,  
     6-107 to 6-108, 6-111, 6-114  
 Simple 5-13, 6-26, 6-28 to 6-29, 6-41 to 6-43,  
     6-50 to 6-51, 6-104 to 6-106, 6-115 to 6-117,  
     6-120 to 6-121  
 Subroutine 5-17, 6-49  
 Taken/not-taken cases 4-4, 6-7  
 Unary 5-13, 6-44, 6-46, 6-118 to 6-119  
 Unsigned 5-13, 6-33 to 6-34, 6-38 to 6-39,  
     6-109 to 6-110, 6-112 to 6-113  
 BRCLR instruction 6-45  
 BRN instruction 6-46  
 BRSET instruction 6-47  
 BSET instruction 6-48  
 BSR instruction 4-3, 6-49  
 Bus cycles 6-5  
 Bus structure B-4  
 BVC instruction 6-50  
 BVS instruction 6-51  
 Byte moves 6-144  
 Byte order in memory 2-5  
 Byte-sized instructions 4-4 to 4-5

## C

C status bit 2-5, 6-19 to 6-26, 6-28, 6-33 to 6-34,  
     6-38 to 6-39, 6-54, 6-69, 6-72 to 6-74,  
     6-78 to 6-79, 6-81 to 6-86, 6-95 to 6-98,  
     6-104 to 6-105, 6-109 to 6-110,  
     6-112 to 6-113, 6-131 to 6-140,  
     6-142 to 6-143, 6-168, 6-170 to 6-175,  
     6-179 to 6-182, 6-193 to 6-195  
 CALL instruction 3-12, 4-3, 5-17, 6-52,  
     10-2 to 10-3, B-16, C-4 to C-5  
 Case statements C-4  
 CBA instruction 6-53  
 Changes in execution flow 4-2 to 4-5,  
     6-102 to 6-103, 6-176 to 6-178, 6-196,  
     7-1 to 7-6  
 CLC instruction 6-54  
 Clear instructions 5-6, 6-56 to 6-58  
 Cleared 1-3  
 CLI instruction 6-55  
 Clock monitor reset 7-3  
 CLR instruction 6-56  
 CLRA instruction 6-57  
 CLRB instruction 6-58  
 CLV instruction 6-59  
 CMPA instruction 6-60

CMPB instruction 6-61  
 Code size B-10  
 COM instruction 6-62  
 COMA instruction 6-63  
 COMB instruction 6-64  
 Compare instructions 5-5, 6-53, 6-60 to 6-61,  
     6-65 to 6-68  
 Complement instructions 5-6, 6-62 to 6-64  
 Computer operating properly monitor 7-3  
 Condition codes instructions 5-21, 6-18,  
     6-54 to 6-55, 6-59, 6-153, 6-156, 6-162,  
     6-182 to 6-184, 6-198, 6-203 to 6-204, B-15  
 Condition codes register 2-1, 2-3, 6-18,  
     6-54 to 6-55, 6-59, 6-90, 6-128, 6-153, 6-156,  
     6-162, 6-177, 6-183 to 6-185, 6-198,  
     6-203 to 6-204, 6-206 to 6-208, C-4  
 C status bit 2-5, 6-19 to 6-26, 6-28, 6-33 to 6-34,  
     6-38 to 6-39, 6-54, 6-69, 6-72 to 6-74,  
     6-78 to 6-79, 6-81 to 6-86, 6-95 to 6-98,  
     6-104 to 6-105, 6-109 to 6-110,  
     6-112 to 6-113, 6-131 to 6-140,  
     6-142 to 6-143, 6-168, 6-170 to 6-175,  
     6-179 to 6-182, 6-193 to 6-195  
 H status bit 2-4, 6-8, 6-11 to 6-14, 6-69  
 I mask bit 2-4, 6-18, 6-55, 6-183, 6-196, 6-205,  
     6-213, 7-2, 7-4  
 Manipulation 5-21, 6-18, 6-54 to 6-55, 6-59,  
     6-153, 6-182 to 6-184, 6-198, 6-204  
 N status bit 2-4, 6-41, 6-43, 6-115, 6-117  
 S control bit 2-3, 6-189  
 Stacking 6-156, 6-162  
 V status bit 2-4, 6-50 to 6-51, 6-59,  
     6-120 to 6-121, 6-166 to 6-169, 6-184  
 X mask bit 2-3, 6-90, 6-162, 6-177, 6-189, 6-198,  
     6-203, 6-213, 7-2, 7-4  
 Z status bit 2-4, 6-29, 6-42, 6-81 to 6-84,  
     6-100 to 6-101, 6-106, 6-116,  
     6-139 to 6-140, 6-142 to 6-143  
 Conditional 16-bit read cycle 6-7  
 Conditional 8-bit read cycle 6-7  
 Conditional 8-bit write cycle 6-7  
 Conserving power 5-21, 6-189  
 Constant indirect indexed addressing mode 3-7  
 Constant offset indexed addressing mode  
     3-6 to 3-7  
 Conventions 1-3  
 COP reset 7-3  
 CPD instruction 6-65  
 CPS instruction 6-66  
 CPU wait 6-213  
 CPX instruction 6-67  
 CPY instruction 6-68  
 Cycle code letters 6-5  
 Cycle counts B-9

Cycle-by-cycle operation 6-5

## D

DAA instruction 6-69  
DATA mnemonic 1-3  
Data types 2-5  
DBEQ instruction 6-70, A-25  
DBNE instruction 6-71, A-25  
DEC instruction 6-72  
DECA instruction 6-73  
DECB instruction 6-74  
Decrement instructions 5-4, 6-72 to 6-77  
Defuzzification 9-6, 9-22 to 9-24, 9-26, 9-29  
DES instruction 6-75  
DEX instruction 6-76  
DEY instruction 6-77  
Direct addressing mode 3-3  
Division instructions 5-7  
    16-bit fractional 6-91  
    16-bit integer 6-94 to 6-95  
    32-bit extended 6-78 to 6-79  
Divison instructions C-3

## E

EDIV instruction 6-78  
EDIVS instruction 6-79  
Effective address 3-2, 3-5, 6-128 to 6-130  
EMACS instruction 5-11, 6-80, 9-1, 9-29  
EMAXD 6-81  
EMAXD instruction 6-81  
EMAXM instruction 6-82, 9-1  
EMIND instruction 6-83, 9-1  
EMINM instruction 6-84  
EMUL instruction 6-85  
EMULS instruction 6-86  
Enabling maskable interrupts 2-4  
EORA instruction 6-87  
EORB instruction 6-88  
ETBL instruction 5-12, 6-89, 9-1  
Even bytes 2-5  
Exceptions 4-3, 7-1  
    Interrupts 7-3  
    Maskable interrupts 7-1, 7-4 to 7-5  
    Non-maskable interrupts 7-1, 7-4  
    Priority 7-2  
    Processing flow 7-6  
    Resets 7-1 to 7-3  
    Software interrupts 5-18, 6-196, 7-1, 7-6  
    Unimplemented opcode trap 7-1 to 7-2, 7-5  
    Vectors 7-1, 7-6  
Exchange instructions 5-2, 6-90, 6-215 to 6-216,  
    B-11, B-13  
    Postbyte encoding A-24

Execution cycles 6-5

    Conditional 16-bit read 6-7  
    Conditional 8-bit read 6-7  
    Conditional 8-bit write 6-7  
    Free 6-5  
    Optional 4-4 to 4-5, 6-6  
    Program word access 6-6  
    Read indirect pointer 6-5  
    Read indirect PPAGE value 6-5  
    Read PPAGE 6-5  
    Read 16-bit data 6-6  
    Read 8-bit data 6-6  
    Stack 16-bit data 6-6  
    Stack 8-bit data 6-6  
    Unstack 16-bit data 6-7  
    Unstack 8-bit data 6-6  
    Vector fetch 6-7  
    Write PPAGE 6-5  
    Write 16-bit data 6-6  
    Write 8-bit data 6-6

Execution time 6-5

EXG instruction 6-90

Expanded memory 3-12, 4-3, 10-1, B-16,  
    C-4 to C-5

    Addressing modes 3-12, 10-4 to 10-6  
    Bank switching 3-12, 10-1, 10-3 to 10-6  
    Chip-select circuits 10-4  
    Instructions 3-12, 5-17, 6-52, 6-176, 10-2 to 10-3  
    Overlay windows 10-1, 10-3 to 10-6  
    Page registers 3-12, 10-1, 10-4 to 10-6  
    Registers 10-5 to 10-6  
    Subroutines 5-17, 10-2, C-4 to C-5

Extended addressing mode 3-3

Extended division 5-7

Extension byte 3-5

External interrupts 7-5

External queue reconstruction 8-1

External reset 7-3

## F

Fast math B-9

FDIV instruction 6-91

Fractional division 5-7

Frame pointer C-2 to C-3

Free cycle 6-5

Fuzzy logic 9-1

    Antecedants 9-5

    Consequents 9-5

    Custom programming 9-26

    Defuzzification 5-9, 9-6, 9-22 to 9-24, 9-26, 9-29

    Fuzzification 5-9, 9-3, 9-26

    Inference kernel 5-9, 9-2, 9-7

    Inputs 5-9, 9-30

- Instructions 5-9, 6-141, 6-166, 6-168, 6-214, 9-1, 9-9, 9-13 to 9-14, 9-17 to 9-19, 9-22, B-14
- Interrupts 9-20, 9-23 to 9-24, 9-26
- Knowledge base 9-2, 9-5
- Membership functions 5-9, 6-141, 9-1 to 9-3, 9-9 to 9-13, 9-26 to 9-27
- Outputs 5-9, 9-30
- Rule evaluation 5-9, 6-166, 6-168, 9-1, 9-5, 9-13 to 9-15, 9-17 to 9-20, 9-22, 9-29
- Rules 9-2, 9-5
- Sets 9-2
- Tabular membership functions 5-12, 9-26
- Weighted average 5-9, 6-214, 9-1, 9-6, 9-22 to 9-24, 9-26

## G

- General purpose accumulators 2-1

## H

- H status bit 2-4, 6-8, 6-11 to 6-14, 6-69
- High-level language C-1, C-3
  - Addressing modes C-1, C-3 to C-4
  - Condition codes register C-4
  - Expanded memory C-4 to C-5
  - Instructions C-1
  - Loop primitives C-3
  - Stack C-1 to C-2

## I

- I mask bit 2-4, 6-18, 6-55, 6-183, 6-196, 6-205, 6-213, 7-2
- IBEQ instruction 6-92, A-25
- IBNE A-25
- IBNE instruction 6-93
- IDIV instruction 6-94
- IDIVS instruction 6-95, C-3
- Immediate addressing mode 3-2
- INC instruction 6-96
- INCA instruction 6-97
- INCB instruction 6-98
- Increment instructions 5-4, 6-96 to 6-101
- Index calculation instructions 5-20, 6-9 to 6-10, 6-76 to 6-77, 6-100 to 6-101, 6-129 to 6-130, B-11
- Index manipulation instructions 5-19, 6-67 to 6-68, 6-90, 6-126 to 6-127, 6-158 to 6-159, 6-164 to 6-165, 6-191 to 6-192, 6-203, 6-209 to 6-212, 6-215 to 6-216
- Index registers 2-1 to 2-2, 5-19, C-2
  - X 3-5, 6-9, 6-67, 6-70 to 6-71, 6-76, 6-90 to 6-95, 6-100, 6-126, 6-128 to 6-130, 6-158, 6-164, 6-166, 6-168, 6-177, 6-185, 6-191, 6-196, 6-200 to 6-203, 6-209, 6-211, 6-215
  - Y 3-5, 6-10, 6-68, 6-70 to 6-71, 6-77 to 6-80,

- 6-85 to 6-86, 6-90, 6-92 to 6-93, 6-101, 6-127 to 6-130, 6-159, 6-165 to 6-166, 6-168, 6-177, 6-185, 6-192, 6-196, 6-200 to 6-203, 6-210, 6-212, 6-216
- Indexed addressing modes 2-2, 3-5, A-22, B-6 to B-9
  - Accumulator direct 3-9
  - Accumulator offset 3-9
  - Automatic indexing 3-8
  - Base index register 3-6, 3-10
  - Extension byte 3-5
  - Postbyte 3-5
  - Postbyte encoding 3-5, A-22
  - 16-bit constant indirect 3-7
  - 16-bit constant offset 3-7
  - 5-bit constant offset 3-6
  - 9-bit constant offset 3-7
- Inference kernel, fuzzy logic 9-7
- Inherent addressing mode 3-2
- INS instruction 6-99
- Instruction queue 1-1, 2-5, 4-1, 8-1, B-4
  - Alignment 4-1
  - Buffer 4-1
  - Debugging 8-1
  - Movement cycles 4-2
  - Reconstruction 8-1, 8-3, 8-5
  - Stages 4-1, 8-1
  - Status registers 8-4 to 8-5
  - Status signals 4-1, 8-1 to 8-3, 8-5 to 8-6
- Instruction set A-2
- Integer division 5-7
- Interrupt instructions 5-18
- Interrupts 7-3
  - Enabling and disabling 2-3 to 2-4, 6-55, 6-183, 7-2
  - External 7-5
  - I mask bit 2-4, 6-55, 6-183, 6-196, 6-213, 7-4
  - Instructions 5-18, 6-55, 6-177, 6-183, 6-196, 6-205
  - Low-power stop 5-21, 6-189
  - Maskable 2-4, 7-4
  - Non-maskable 2-3, 7-2, 7-4
  - Recognition 7-4
  - Return 2-4, 5-18, 6-177, 7-5
  - Service routines 7-4
  - Software 5-18, 6-196, 7-1, 7-6
  - Stacking 7-4
  - Vectors 7-3
  - Wait instruction 5-21, 6-213
  - X mask bit 2-3, 6-189, 6-213, 7-4
- INX instruction 6-100
- INY instruction 6-101

## J

JMP instruction 4-5, 6-102  
JSR instruction 4-3, 6-103  
Jump instructions 5-17  
Jumps 4-5

## K

Knowledge base 9-2

## L

LBCC instruction 6-104  
LBCS instruction 6-105  
LBEQ instruction 6-106  
LBGE instruction 6-107  
LBGT instruction 6-108  
LBHI instruction 6-109  
LBHS instruction 6-110  
LBLE instruction 6-111  
LBLO instruction 6-112  
LBLS instruction 6-113  
LBLT instruction 6-114  
LBMI instruction 6-115  
LBNE instruction 6-116  
LBPL instruction 6-117  
LBRA instruction 6-118  
LBRN instruction 6-119  
LBVC instruction 6-120  
LBVS instruction 6-121  
LDAA instruction 6-122  
LDAB instruction 6-123  
LDD instruction 6-124  
LDS instruction 6-125  
LDX instruction 6-126  
LDY instruction 6-127  
LEAS instruction 6-128, C-2, C-4  
Least significant byte 1-3  
Least significant word 1-3  
LEAX instruction 6-129, C-4  
LEAY instruction 6-130, C-4  
Legal label 6-3  
Literal expression 6-3  
Load instructions 5-1, 6-122 to 6-130  
Logic level one 1-3  
Logic level zero 1-3  
Loop primitive instructions 4-5, 6-70 to 6-71,  
6-92 to 6-93, 6-200, 6-202, A-25, B-13, C-3  
Offset values 5-16  
Postbyte encoding A-25  
Low-power stop 5-21, 6-189  
LSL instruction 6-131  
LSL mnemonics 5-8  
LSLA instruction 6-132  
LSLB instruction 6-133

LSLD instruction 6-134  
LSR instruction 6-135  
LSRA instruction 6-136  
LSRB instruction 6-137  
LSRD instruction 6-138

## M

Maskable interrupts 7-1, 7-4  
MAXA instruction 6-139  
Maximum instructions 5-11, B-14  
16-bit 6-81 to 6-82  
8-bit 6-139 to 6-140  
MAXM instruction 6-140, 9-1  
MEM instruction 5-9, 6-141, 9-1, 9-9 to 9-13  
Membership functions 9-2  
Memory and addressing symbols 1-2  
Memory expansion  
Addressing 10-7  
Bank switching 10-7  
Overlay windows 10-7  
Page registers 10-3, 10-7  
MINA instruction 6-142, 9-1  
Minimum instructions 5-11, B-14  
16-bit 6-83 to 6-84  
8-bit 6-142 to 6-143  
MINM instruction 6-143  
Misaligned instructions 4-4 to 4-5  
Mnemonic 1-3  
Mnemonic ranges 1-3  
Most significant byte 1-3  
Most significant word 1-3  
MOVB instruction 6-144  
Move instructions 5-3, 6-144 to 6-145, B-10, B-13  
Destination 3-10  
Multiple addressing modes 3-10  
PC relative addressing 3-10  
Reference index register 3-10  
Source 3-10  
MOVW instruction 6-145  
MUL instruction 6-146  
Multiple addressing modes  
Bit manipulation instructions 3-11, 6-27, 6-48  
Move instructions 3-10, 6-144 to 6-145  
Multiplication instructions 5-7  
16-bit 6-85 to 6-86  
8-bit 6-146  
Multiply and accumulate instructions 5-11, 6-80,  
6-214  
M68HC11 compatibility 3-2, B-1  
M68HC11 instruction mnemonics B-1

## N

N status bit 2-4, 6-41, 6-43, 6-115, 6-117

NEG instruction 6-147  
 NEGA instruction 6-148  
 Negate instructions 5-6, 6-147 to 6-149  
 Negated 1-3  
 Negative integers 2-5  
 NEGB instruction 6-149  
 Non-maskable interrupts 7-1 to 7-2, 7-4  
 NOP instruction 5-22, 6-150  
 Notation

- Branch taken/not taken 6-7
- Changes in CCR bits 6-2
- Cycle-by-cycle operation 6-5
- Memory and addressing 1-2
- Object code 6-2
- Operators 1-3
- Source forms 6-3
- System resources 1-2

Null operation instruction 5-22, 6-150  
 Numeric range of branch offsets 3-4

## O

Object code notation 6-2  
 Odd bytes 2-5  
 Opcodes B-2, B-9  
 Map A-20  
 Operators 1-3  
 Optional cycles 4-4 to 4-5, 6-6  
 ORAA instruction 6-151  
 ORAB instruction 6-152  
 ORCC instruction 6-153  
 Orthogonality C-5

## P

Pointer calculation instructions 5-20,  
 6-128 to 6-130  
 Pointers C-4  
 Postbyte 3-5, 6-90, 6-185, 6-203  
 Postbyte encoding  
 Exchange instructions A-24  
 Indexed addressing modes A-22  
 Loop primitive instruction A-25  
 Transfer instructions A-24  
 Post-decrement indexed addressing mode 3-8  
 Post-increment indexed addressing mode 3-8  
 Power conservation 5-21, 6-189, 6-213  
 Power-on reset 7-3  
 Pre-decrement indexed addressing mode 3-8  
 Pre-increment indexed addressing mode 3-8  
 Priority, exception 7-2  
 Program counter 2-1 to 2-2, 3-5, 6-31, 6-49, 6-52,  
 6-103, 6-128 to 6-130, 6-144 to 6-145, 6-150,  
 6-177 to 6-178, 6-196, 6-201, 6-205  
 Program word access cycle 6-6  
 Programming model 1-1, 2-1, B-3

Pseudo-non-maskable interrupt 7-2

PSHA instruction 6-154  
 PSHB instruction 6-155  
 PSHC instruction 6-156  
 PSHD instruction 6-157  
 PSHX instruction 6-158  
 PSHY instruction 6-159  
 PULA instruction 6-160  
 PULB instruction 6-161  
 PULC instruction 6-162  
 PULD instruction 6-163, C-2  
 Pull instructions C-5  
 PULX instruction 6-164  
 PULY instruction 6-165  
 Push instructions C-5  
 PUSH instruction C-2

## R

Range of mnemonics 1-3  
 Read indirect PPAGE cycle 6-5  
 Read PPAGE cycle 6-5  
 Read 8-bit data cycle 6-6  
 Read16-bit data cycle 6-6  
 Register designators 6-3  
 Relative addressing mode 3-4  
 Relative offset 3-4  
 Resets 7-1 to 7-2  
 Clock monitor 7-3  
 COP 7-3  
 External 7-3  
 Power-on 7-3  
 REV instruction 5-9, 6-166, 9-1, 9-5, 9-13 to 9-15,  
 9-17 to 9-20, 9-22, 9-29  
 REVW instruction 5-9, 6-168, 9-1, 9-5,  
 9-13 to 9-15, 9-17 to 9-20, 9-22, 9-29  
 ROL instruction 6-170  
 ROLA instruction 6-171  
 ROLB instruction 6-172  
 ROM, BDM 8-6  
 ROR instruction 6-173  
 RORA instruction 6-174  
 RORB instruction 6-175  
 Rotate instructions 5-8, 6-170 to 6-175  
 RTC instruction 3-12, 4-3, 5-17, 6-176,  
 10-2 to 10-3, B-16, C-4 to C-5  
 RTI instruction 2-4, 5-18, 6-177, 7-5  
 RTS instruction 4-3, 6-178

## S

S control bit 2-3, 6-189  
 SBA instruction 6-179  
 SBCA instruction 6-180  
 SBCB instruction 6-181  
 SEC instruction 6-182

SEI instruction 6-183  
 Set 1-3  
 Setting memory bits 6-48  
 SEV instruction 6-184  
 SEX instruction 5-2, 6-185  
 Shift instructions 5-8  
     Arithmetic 6-19 to 6-25  
     Logical 6-131 to 6-138  
 Sign extension instruction 6-185  
 Signed branches 5-13  
 Signed integers 2-5  
 Signed multiplication 5-7  
 Sign-extension instruction 5-2, C-1  
 Simple branches 5-13  
 Software interrupts 6-196, 7-1  
 Source code compatibility 1-1, B-1  
 Source form notation 6-3  
 Specific mnemonic 1-3  
 STAA instruction 6-186  
 STAB instruction 6-187  
 Stack 2-2, B-5 to B-6  
     Interrupts 6-177, 6-196  
     Stop and wait 6-189, 6-213  
     Subroutines 6-49, 6-52, 6-103, 6-176, 6-178  
     Traps 6-205  
 Stack operation instructions 5-20, 6-154 to 6-165  
 Stack pointer 2-1 to 2-2, 3-5, 6-49, 6-52, 6-66,  
     6-70 to 6-71, 6-75, 6-90, 6-92 to 6-93, 6-99,  
     6-103, 6-125, 6-128 to 6-130, 6-155 to 6-165,  
     6-178, 6-185, 6-190, 6-200 to 6-203,  
     6-209 to 6-212, C-1  
     Initialization 2-2  
     Manipulation 5-20, 6-66, 6-75, 6-99, 6-125,  
         6-128, 6-154 to 6-155, 6-190, 6-209 to 6-212  
     Stacking order 2-2, B-5  
 Stack pointer instructions 5-20, 6-66, 6-75, 6-99,  
     6-125, 6-128, 6-190, 6-203, 6-209 to 6-212,  
     B-15, C-1  
 Stack 16-bit data cycle 6-6  
 Stack 8-bit data cycle 6-6  
 Stacking instructions 6-154 to 6-155  
 Standard CPU12 address space 2-5  
 STD instruction 6-188  
 STOP instruction 2-3, 5-21, 6-189  
 Store instructions 5-1, 6-186 to 6-188,  
     6-190 to 6-192  
 STS instruction 6-190  
 STX instruction 6-191  
 STY instruction 6-192  
 SUBA instruction 6-193  
 SUBB instruction 6-194  
 SUBD instruction 6-195  
 Subroutine instructions 5-17

Subroutines 4-3, 6-103, C-4 to C-5  
     Expanded memory 4-3, 5-17, 6-52, 6-176  
     Instructions 5-17, 6-49, 6-103, C-4 to C-5  
     Return 6-176, 6-178  
 Subtraction instructions 5-3, 6-179 to 6-181,  
     6-193 to 6-195  
 SWI instruction 5-18, 6-196, 7-6  
 Switch statements C-4  
 Symbols and notation 1-2

## T

TAB instruction 6-197  
 Table interpolation instructions 5-12, 6-89, 6-201,  
     B-15  
 Tabular membership functions 9-26 to 9-27  
 TAP instruction 6-198  
 TBA instruction 6-199  
 TBEQ instruction 6-200, A-25  
 TBL instruction 5-12, 6-201, 9-1, 9-26 to 9-27  
 TBNE instruction 6-202, A-25  
 Termination of interrupt service routines 5-18,  
     6-177, 7-5  
 Termination of subroutines 6-176, 6-178  
 Test instructions 5-5, 6-35 to 6-36, 6-200, 6-202,  
     6-206 to 6-208  
 TFR instruction 6-185, 6-198, 6-203 to 6-204,  
     6-209 to 6-212  
 TPA instruction 6-204  
 Transfer and exchange instructions C-1  
 Transfer instructions 5-2, 6-197 to 6-199,  
     6-203 to 6-204, 6-209 to 6-212, B-11, B-13  
     Postbyte encoding A-24  
 TRAP instruction 5-18, 6-205, 7-5  
 TST 6-206  
 TST instruction 6-206  
 TSTA instruction 6-207  
 TSTB instruction 6-208  
 TSX instruction 6-209  
 TSY instruction 6-210  
 Twos-complement form 2-5  
 TXS instruction 6-211  
 Types of instructions  
     Addition and Subtraction 5-3  
     Background and null 5-22  
     Binary-coded decimal 5-4  
     Bit test and manipulation 5-7  
     Boolean logic 5-6  
     Branch 5-13  
     Clear, complement, and negate 5-6  
     Compare and test 5-5  
     Condition code 5-21  
     Decrement and increment 5-4  
     Fuzzy logic 5-9



- Index manipulation 5-19
- Interrupt 5-18
- Jump and subroutine 5-17
- Load and store 5-1
- Loop primitives 5-16
- Maximum and minimum 5-11
- Move 5-3
- Multiplication and division 5-7
- Multiply and accumulate 5-11
- Pointer and index calculation 5-20
- Shift and rotate 5-8
- Sign extension 5-2
- Stacking 5-20
- Stop and wait 5-21
- Table interpolation 5-12
- Transfer and exchange 5-2
- TYS instruction 6-212

## U

- Unary branches 5-13
- Unimplemented opcode trap 5-18, 6-205, 7-1 to 7-2
- Unsigned branches 5-13
- Unsigned multiplication 5-7
- Unstack 16-bit data cycle 6-7
- Unstack 8-bit data cycle 6-6
- Unweighted rule evaluation 6-166, 9-5, 9-13 to 9-15, 9-17 to 9-20, 9-22, 9-29

## V

- V status bit 2-4, 6-50 to 6-51, 6-59, 6-120 to 6-121, 6-166 to 6-169, 6-184
- Vector fetch cycle 6-7
- Vectors, exception 7-1, 7-6

## W

- WAI instruction 5-21, 6-213
- WAV instruction 5-9, 5-11, 6-214, 9-1, 9-6, 9-22 to 9-24, 9-26, 9-29
- Wavr pseudoinstruction 9-23 to 9-24, 9-26
- Weighted average 6-214
- Weighted rule evaluation 6-168, 9-5, 9-13 to 9-15, 9-17 to 9-20, 9-22, 9-29
- Word moves 6-145
- Write PPAGE cycle 6-5
- Write 16-bit data cycle 6-6
- Write 8-bit data cycle 6-6

## X

- X mask bit 2-3, 6-90, 6-162, 6-177, 6-189, 6-198, 6-203, 6-213
- XGDX instruction 6-215
- XGDY instruction 6-216

## Z

- Z status bit 2-4, 6-29, 6-42, 6-81 to 6-84, 6-100 to 6-101, 6-106, 6-116, 6-139 to 6-140, 6-142 to 6-143
- Zero-page addressing 3-3

## SUMMARY OF CHANGES

This is a complete revision and reprint. All known errors in the publication have been corrected. The following summary lists significant changes.

Page	Change
3-6	Additional information provided in Table 3-2.
3-9	Changed paragraph 3.8.6 to indicate accumulator offset is an unsigned value.
4-5	Changed paragraph 4.3.3.4 to show that both taken and not taken cases for loop primitives use the same number of P cycles.
5-18	Table 5-22, operation sequence of RTI instruction modified to match sequence in Sec. 6.
6-3 and 6-4	Removed spurious letter "e" from "opr" source forms.
6-11 to 6-14	Added overbars to terms in Boolean formulae for ADCA, ADCB, ADDA, and ADDB.
6-27	Modified V bit description of condition code register.
6-70, 6-71, 6-92, 6-93, 6-200 and 6-202	Corrected access details for loop primitives to show that taken and not taken cases both use three P cycles.
6-78, 6-79, 6-94	Correction in descriptions for EDIV, EDIVS, and IDIV: "dividend" is divided by "divisor."
6-78	Comment removed in EDIV description regarding C status bit.
6-81, 6-82, 6-83, 6-84, 6-139, 6-140, 6-142, 6-143, 6-193, 6-194, 6-195	In condition code C bit description of EMAXD, EMAXM, EMIND, EMINM, MAXA, MAXM, MINA, MINM, SUBA, SUBB and SUBD, two occurrences of the word "absolute" have been removed.
6-148	Overbar added to term in NEGA operation description.
6-167	Corrected access detail for REV instruction.
6-177	Corrected operation sequence for RTI instruction.
6-189	Corrected operation sequence for STOP instruction. Also, fourth paragraph of description modified so as to not indicate that SP is changed.
6-196	Condition code register corrected; status bit I is set (1) following the SWI instruction.
6-213	Corrected operation sequence for WAI instruction.
6-214	Corrected access detail for WAV instruction.
8-7	Section 8.4.2, second paragraph, time-out of 256 E clock cycles is changed to 512 E clock cycles. Fourth paragraph, "Nine target E-cycles later," is now "Ten target E-cycles later."
8-8	Figure 8-2, nine cycle reference is changed to ten cycles; art is modified accordingly.
8-10	Table 8-2, command order changed, footnote explanations added, ENTER_TAG_MODE command deleted.
8-12	Section 8.4.4.1, reset conditions for STATUS register corrected. ITF bit name is changed to ENTAG, Instruction Tagging Enable.
9-16 and 9-21	Corrected flow arrow and font substitution errors in Figures 9-9 and 9-10.
9-24	Changed paragraph 9.6.3. to reflect a three-cycle delay rather than a four-cycle delay.
9-25	Corrected flow arrow error and removed cycle 10.1 Figure 9-11.
9-28	Figure 9-12, Corrected inappropriate line break in code.
B-10	Table B-3, last row (EMACS) math operation corrected and two occurrences of "per iteration" removed.
B-13	Section B.7.2, first sentence, "six transfer instructions" is now "eight transfer instructions."
General	Minor grammatical and typographic corrections to improve consistency and presentation. New index markers.



